Miikka Salomaa

Karttapohjainen sopimuksenhallintajärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

23.01.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Miikka Salomaa Karttapohjainen sopimuksenhallintajärjestelmä 40 sivua + 5 liitettä 23.01.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	lehtori Peter Hjort diplomi-insinööri Olli Alanko

Insinöörityönä suunniteltiin ja toteutettiin karttapohjainen sopimuksenhallintajärjestelmä HTML5- ja Java-teknologioita hyödyntäen. Sopimuksenhallintajärjestelmä auttaa sopimuksen osapuolia kommunikoimaan saumattomasti keskenään ja näkemään helposti sopimuksen ydinkohdat, sekä maantieteelliset rajat kartalla.

Karttapohjainen sopimuksenhallintajärjestelmän suunnittelun ja toteutuksen tavoitteena oli tehdä luotettava työkalu loppukäyttäjille ja pysyä aikataulussa. Järjestelmästä tuli suunnitella helposti käytettävä ja ehjä kokonaisuus ottaen huomioon uusien ominaisuuksien lisääminen järjestelmään myöhemmin.

Karttapohjainen sopimuksenhallintajärjestelmä toteutettiin Java-, jQuery-, OpenLayers- ja HTML5-teknologioilla ja olio-ohjelmoinnin suunnittelumenetelmiä hyödyntäen. Java-ohjelmointikielellä toteutettiin palvelinpuolen ohjelmisto ja selainpuolen ohjelmisto JavaScript, HTML ja CSS -ohjelmointikielillä. Sovelluksen toteutuksessa hyödynnettiin insinöörityön tilaajan Geometrix Oy:n aikaisempien sovelluksien arkkitehtuuria.

Insinöörityössä kuvattu sovellus saatiin valmiiksi aikataulussa kaikilla oleellisimmilla toiminnoilla. Sovellusta käytetään pilottihankkeessa talvihoidon laadun tarkkailussa, siksi järjestelmä täytyi saada tuotantoon ennen talven alkamista. Karttapohjainen sopimuksenhallintajärjestelmän käyttäjät olivat tyytyväisiä tuotteeseen ja se täytti asiakkaan odotukset toimivasta työkalusta laadun tarkkailuun.

Avainsanat	Java, OpenLayers, paikkatieto, JavaScript

Author Title Number of Pages Date	Miikka Salomaa Map-based contract management system 40 pages + 5 appendices 23 January 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer Olli Alanko, M.Sc., Project manager

The thesis covers the design and development of a map-based contract management system with Java and HTML5 technologies. The system helps the subscriber and the contractor to communicate and to see contract details and geographical boundaries on a map.

The goal was to produce a high quality and reliable tool with good usability and to keep the time limit of the project plan. It was important to consider later development in design and make it easy to add later new features to it.

The map-based contract management system was developed with Java, jQuery and HTML5 technologies. The design applies object oriented designing patterns. The server side application was developed with the Java programming language and the client side with the JavaScript, CSS and HTML5 programming languages. Architectures from the previous applications of Geometrix Oy were used to help the development of this application.

The application was released on time with all necessary features. Helsinki city uses this application to observe the quality of winter maintenance; therefore, it was important to release this system before winter. End users were satisfied with this product and the application met the requirements of the subscriber.

Keywords	Java, OpenLayers, Geographical information, JavaScript

Sisällys

Liitteet

Liite 1. Sopimuksen tilan katselun käyttötapaus

Liite 2. Sovelluksen integraatiotestaus

Lyhenteet

AJAX	Asynchronous JavaScript And XML; JavaScript-tekniikka sivun päivittämiseen ilman sivun uudelleen lataamista.
API	Application Programming Interface; ohjelmointirajapinta.
CGI	Common Gate Interface; Web-ympäristön tekniikka.
DTO	Data Transfer Object; Java-olio tiedon välittämiseen.
FTP	File Transfer Protocol; protokolla tiedostojen siirtoon internetissä.
HTML	Hypertext Markup Language; hypertekstin kuvauskieli web-sivujen näyttämiseen.
HTTP	Hypertext Transfer Protocol; protokolla tiedon välittämiseen internetissä.
IDE	Integrated Development Environment; kehitysympäristö.
JDBC	Java Database Connectivity; Java-ohjelmointikielen tietokantarajapinta.
JNDI	Java Naming And Directory Interface; Java-ohjelmointikielen rajapinta nimeämis- ja tiedostorakenteille.
JSON	JavaScript Object Notation; tiedonsiirtomuoto

JSP	JavaServer Pages Technology; interaktiivisten web-sivujen kehittämisteknologia.
JSTL	JavaServer Pages Standard Tag Library; JSP-sivujen lausekekieli.
MVC	Model View Controller; ohjelmointiarkkitehtuuri.
POM	Project Object Model; Mavenin konfiguraatiotiedosto.
SQL	Structured Query Language; tietokantojen kyselykieli.
SSH	Secure Shell; suojattu etäyhteys.
SVN	Subversion; versionhallintaohjelma.
REST	Representational State Transfer; arkkitehtuuri ohjelmointirajapintojen tekemiseen.
URL	Uniform Resource Identifier; merkkijono, jota käytetään osoittamaan resurssin sijaintia verkossa.
WFS	Web Feature Service; protokolla karttakohteiden kyselyyn.
WMS	Web Map Service; protokolla karttakuvien kyselyyn.
XML	Extensible Markup Language; merkkikielistandardi.

1 Johdanto

Insinöörityö tehdään Geometrix Oy:lle, joka on paikkatietoon erikoistunut ohjelmistoyritys. Geometrix on toteuttanut paikkatietoon perustuvia sovelluksia, jotka muun muassa helpottavat työnohjausta. Geometrix hyödyntää sovelluksissa mobiililaitteita, joilla on helppo tehdä kenttäraportteja ja luoda omaisuusrekisterejä. Tämä insinöörityö hyödyntää joitain Geometrixin valmiita sovelluksia osana tiedonkeruuta.

Insinöörityö on pilottisovellus, joka toteutettiin syksyllä 2013. Pilottia testattiin Helsingin kaupungin rakennusviraston ja urakoitsijan välisessä sopimuksessa talvihoidon osalta Kontulan yhteistoimintaurakassa.

Sopimuksen osapuolien (tilaajan ja urakoitsijan) saumaton kommunikointi on tärkeää kummankin osapuolen kannalta. Saumaton kommunikointi takaa osapuolien käsityksen sopimukseen liittyvistä ehdoista ja takaa sopimuksen ehtojen täyttymisen. Tässä insinöörityössä sopimuksella tarkoitetaan tilaajan ja urakoitsijan välistä sopimusta, johon kuuluu urakan ajankohta, alue, tilanne ja laatu. Osapuolien yhteinen ymmärrys sopimuksen ehdoista takaa laadukkaan työn ja käsityksen siitä, milloin urakka on valmis. Internetin tuoma kommunikoinnin esteettömyys mahdollistaa sopimuksen osapuolien saumattoman kommunikoinnin.

Sovellus näyttää sopimuksen osapuolille sopimuksen tämän hetkisen tilan yhdellä silmäyksellä. Sovelluksesta käyttäjä näkee häneen liitettyjen sopimusten maantieteelliset rajat, viimeisimmät muutokset ja työn laadun. Työn laatu lasketaan sopimuksien maantieteellisten rajojen sisältä tehdystä laatupoikkeamista. Työn laatu ilmoitetaan käyttäjälle numeroarvolla ja värikoodilla. Käyttäjät voivat itse lisätä järjestelmään organisaatioita ja urakoitsijoita, sekä yhdistää niitä sopimuksiin. Sopimuksiin voidaan lisätä tuotteita ja tuotekortteja jotka liittyvät sopimuksen ehtoihin.

Insinöörityössä kuvataan vaiheittain karttapohjaisen sopimuksenhallintajärjestelmän kehitystä. Aluksi tutkitaan mahdollisia teknologioita ja tekniikoita, joilla sovellus voitaisiin kehittää. Myöhemmin tehdään teknologiavalinnat ja perustellaan niiden valitsemista. Lopuksi kuvataan tarkasti sovelluksen eri kehitysvaiheita aina käyttöliittymästä olio-ohjelmoinnin suunnittelumenetelmiin. Insinöörityö on paikkatietoon perustuva sovellus ja siksi karttakomponentti on tärkeässä roolissa. Karttakomponentin kehitystä kuvataan tarkasti ja laajasti. Karttakomponentin rakentamisessa käytetään standardiprotokollia,

joilla kehitys onnistuu vaivattomasti ja jotka tekevät siitä yhteensopivan muihin paikkatietoon perustuvien sovelluksien kanssa.

Karttapohjainen sopimuksenhallintajärjestelmä ohjelmoidaan pääosin Java-ohjelmointikielellä ja insinöörityössä kuvataan Java-ohjelmointikielen rajapintoja ja kirjastoja. Näitä ovat esimerkiksi testauksessa käytettävä JUnit ja tietokannan kartoituksessa käytettävä Hibernate.

Insinöörityössä kuvattava sovellus on web-pohjainen sovellus, ja se tehdään hyödyntäen HTML5-tekniikoita, joita ovat CSS3, JavaScript ja HTML. Suurimpana JavaScript-kirjastona käytetään jQuery-kirjastoa ja käyttöliittymän interaktiivisuutta lisätään jQueryUI-kirjastolla. Sovelluksessa käytetään myös muita JavaScript-kirjastoja tukemaan kehitystä ja tekemään siitä mahdollisimman laadukasta. Insinöörityössä käydään läpi sovelluksen käytettävyyttä, pureudutaan muutamaan käytettävyysongelmaan sekä kuvataan, miten ongelmat ratkaistaan. Käyttöliittymää ja sovelluksen logiikkaa havainnollistetaan kuvankaappauksilla ja koodiesimerkeillä.

2 Projektin määrittely

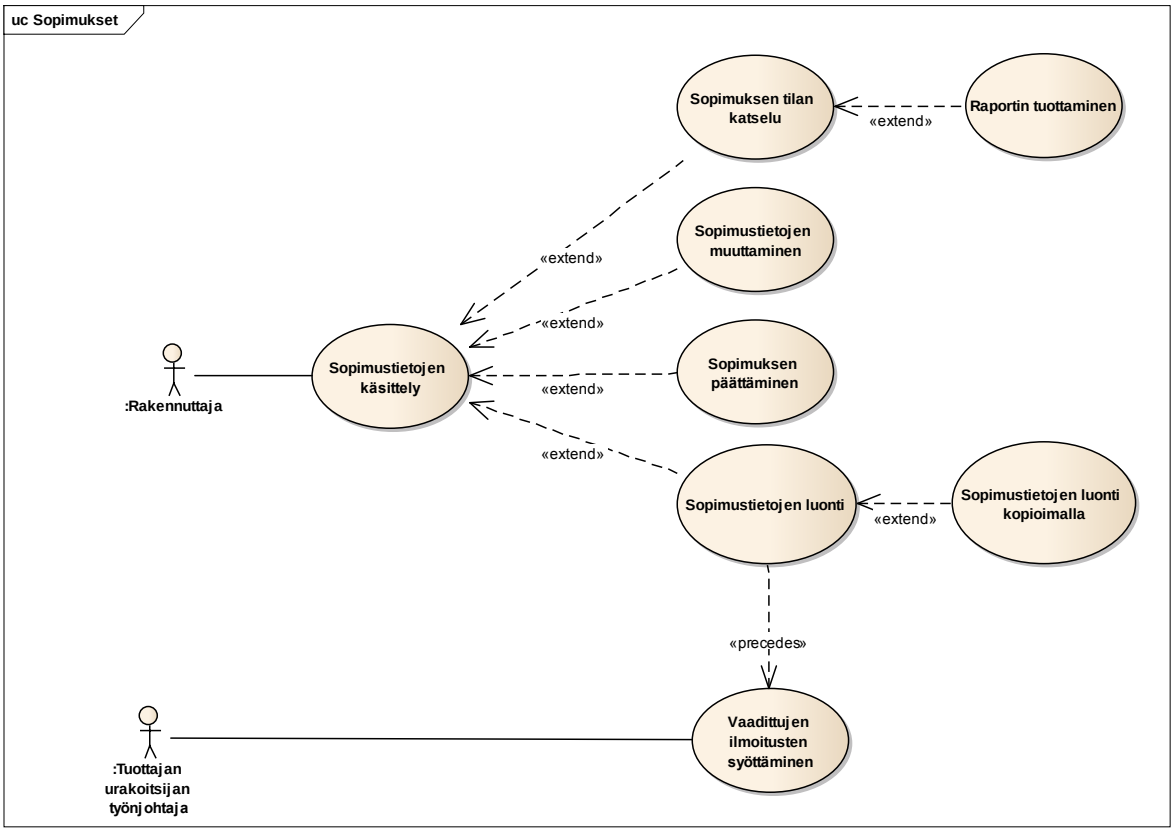
2.1 Käyttötapaukset

Sovelluksen määrittelyn on minulle antanut Geometrix Oy:n toimitusjohtaja ja projektin projektipäällikkö Olli Alanko. Olli Alanko on kirjoittanut sovelluksesta määrittelydokumentteja, joihin kuuluvat esimerkiksi alustava tietomalli, alustavat käyttöliittymäsuunnitelmat ja käyttötapauksia.

Käyttötapaukset auttavat kehittäjää hahmottamaan käyttäjän toimintaa sovelluksen/palvelun kanssa. Käyttötapaus kertoo, mitä käyttäjä tekee ja mihin sovellusta tai palvelua tarvitaan. Yhdessä sovelluksessa on useita käyttötapauksia. Käyttötapaukset kuvataan sanallisesti, jossa kerrotaan esimerkiksi:

- sovelluksen/palvelun nimi

- käyttötapauksen nimi
- suorittajat
- käytettävyysvaatimukset
- esiehdot
- käyttötapauksen varsinainen kuvaus
- poikkeukset
- lopputulos [1]



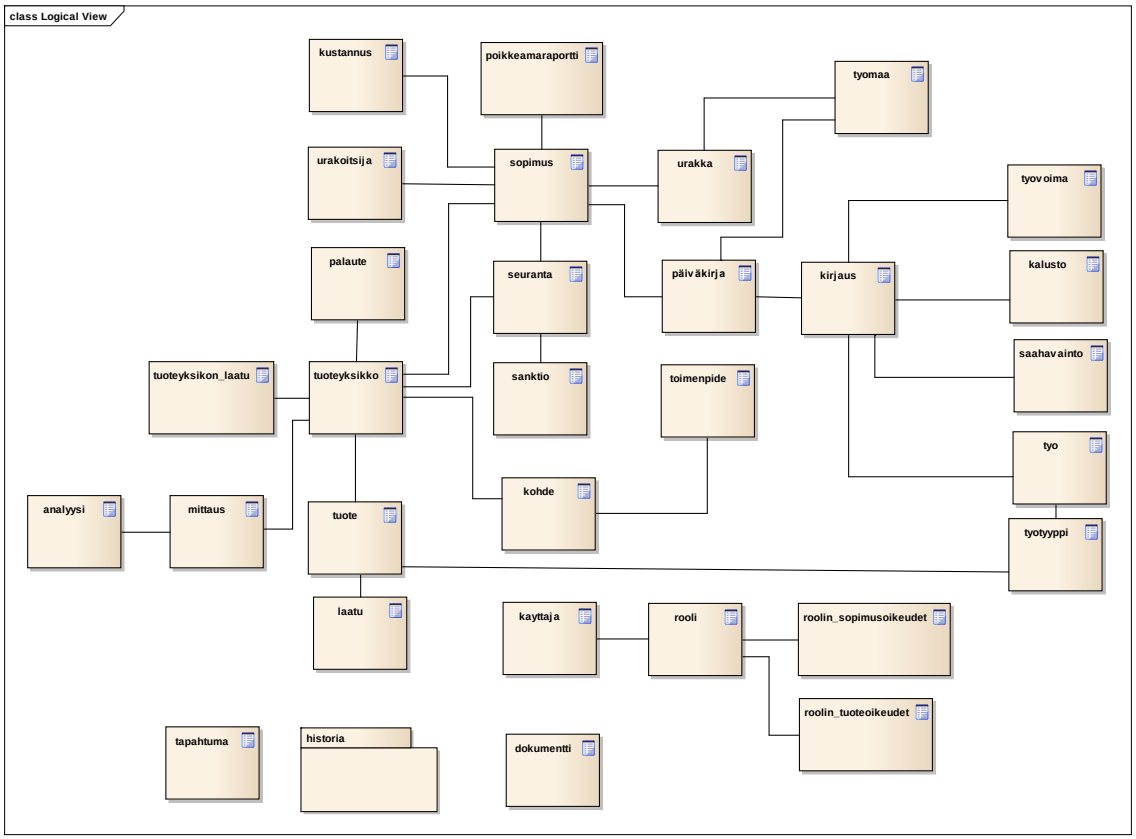
Kuva 1. Sopimuksen käyttötapaukset.

Käyttötapaukset auttavat sovelluskehittäjää muodostamaan kuvan sovelluksen rakenteesta ja sovelluksen yksityisistä toiminnoista. Kuvassa 1 kuvataan sopimuksen osa-

puolien käyttötapauksia. Tämä selventää, mitä toimintoja eri käyttäjät voivat sopimuksen ympärillä tehdä. Liitteessä 1 on sopimuksen katselusta yksityiskohtaisempi käyttötapaus, josta selviävät sopimuksen tietolomakkeen toiminnot. Tällaisen käyttötapauksen pohjalta on hyvä tehdä käyttöliittymäsuunnitelma ja saada tarkempi kuva sovelluksen tapahtumista. Sovelluksen kaikista päätoiminnoista on kirjoitettu vähintään yksi käyttötapaus, josta selviää käyttötapauksen nimi, yhteenveto, tekijät, esiehdot, kuvaus, poikkeukset ja lopputulos [liite 1]. Näiden tarkempien käyttötapauksien pohjalta on hyvä lähteä suunnittelemaan käyttöliittymää ja sovelluksessa navigointia.

2.2 Tietomalli

Tietomalli kuvaa sovelluksen eri osien yhteyksiä ja helpottaa hahmottamaan kokonaisuutta. Abstraktin tietomallin pohjalta on hyvä tehdä tietokantasuunnitelma, johon sovelluksen logiikka perustuu ja tiedot säilötään.



Kuva 2. Sovelluksen tietomalli.

Kuvasta 2 selviävät sopimukseen liittyvät osa-alueet. Sopimukseen liittyy vahvasti urakoitsija, urakka, tuotekortti ja niihin liittyviä erilaisia raportteja. Raportteja voivat olla esimerkiksi työmaapäiväkirjan merkinnät ja poikkeamaraportit. Sopimuksen pohjalta kerätään tietoa ja siitä muodostetaan erilaisia analyyseja.

2.3 Tekniset vaatimukset

Sopimuksenhallintajärjestelmä on selainpohjainen sovellus, joka toimii http-yhteyksien kautta. Sovelluksen on toimittava erityisesti Internet Explorer -selaimessa, koska kyseinen selain on yleisesti käytössä sovelluksen käyttäjillä. Sovelluksessa tulee käyttää Geometrix Oy:ssä hyväksi todettuja teknologioita. Näitä ovat esimerkiksi Java-ohjelmointikieli, PostgreSQL-tietokanta ja Linux-palvelin.

Java on laajalti käytetty ohjelmointikieli. Javalla on suuri kehittäjäkunta, noin 9 miljoonaa. Tästä syystä Javasta löytyy paljon artikkeleita, keskustelua, esimerkkejä ja tukea. Javalla on mahdollista toteuttaa esimerkiksi verkkosovelluksia, pelejä ja mobiilisovelluksia. [2.]

PostgreSQL on avoimeen lähdekoodiin perustuva objekti-relaatiotietokanta. PostgreSQL on arvostettu ja luotettava tietokanta. PostgreSQL:ssä on myös rajapinta Java-ohjelmointikielelle, mikä tukee Javan käyttöä sovelluksen ohjelmointikielenä. [3.]

Linuxin on alun perin kehittänyt Helsingin yliopistossa opiskellut Linus Torvalds. Linux on ilmainen ja avoimen lähdekoodin käyttöjärjestelmän ydin. Linuxiin perustuvia käyttöjärjestelmiä on useimmiten käytetty palvelinratkaisuissa. Linux on luotettavampi ja turvallisempi kuin vastaavat kaupalliset käyttöjärjestelmät. [4, s.1–2.]

3 Toteutuksen suunnittelu

3.1 Tekniikoiden vertailu

Sovellusta suunnitellessa tulee vertailla tekniikoita, jotta sovellus valmistuisi mahdollisimman helposti ja siitä tulisi laadukas. Web-sovellukset on tapana eritellä kahteen osaan: back-endiin ja front-endiin.

Back-end tarkoittaa palvelinpuolen logiikkaa. Back-endissä tapahtuu tiedon hakeminen ja lisääminen tietokantaan ja tiedon palauttaminen sovelluksen front-endiin. Front-end on vastuussa tiedon näyttämisestä käyttäjälle. Back-endissä käytetään Java-ohjelmointikieltä ja PostgreSQL-tietokantaa. Tiedon käsittelyyn Javalla löytyy erilaisia valmiita malleja, joita kutsutan viitekehyksiksi. Viitekehykset tuovat struktuuria arkkitehtuuriin ja nopeuttavat kehityksen aloittamista. Näitä ovat esimerkiksi

- Spring
- Play
- Struts 2
- GWT.

Koska sovelluksesta halutaan mahdollisimman nopea ja vuorovaikutteisempi, sovelluksessa käytetään Ajax-tekniikoita. JavaScript ei ollut aiemmin kehittäjien suuressa suosiossa. Sen käyttö on nykyään yleistynyt Ajax-tekniikan myötä. Ajax-tekniikka mahdollistaa dynaamisten verkkosovellusten luomisen ja yhteyden palvelimeen ilman verkkosivun koko sisällön uudelleenlataamista [5 s.7.].

Ajax-tekniikoiden käyttämistä helpottavia JavaScript-kirjastoja on monia. Yleisesti käytössä oleva ja hyvin dokumentoitu kirjasto on jQuery. jQuery JavaScript-kirjasto on nopea, pieni ja täynnä ominaisuuksia. Sen helppokäyttöinen rajapinta toimii monissa selaimissa. jQueryn helppokäyttöinen rajapinta tekee HTML-dokumentin manipuloinnista, tapahtumien käsittelystä, animoinnista ja Ajaxista yksinkertaisempaa. jQuery on monipuolisuutensa ja laajennettavuutensa ansioista muuttanut miljoonien ihmisten tapaa kirjoittaa JavaScriptiä [6].

Angularjs on uusi JavaScript-kirjasto, joka on nousemassa yhä suosituimmaksi. Angularjs tarjoaa täyden MVC-arkkitehtuurin front-end-viitekehyksen Ajax-sovelluksien toteutukseen, eikä sovellukseen tarvitse lisätä muita erillisiä JavaScript-kirjastoja, jotka hallitsevat esimerkiksi sovelluksen reitityksen.

3.2 Valittavat tekniikat

Sovelluksessa on tarkoitus hyödyntää muita Geometrix Oy:n tekemiä Java web-sovelluksia. Siksi tässä projektissa ei käytetä valmiita tai kaupallisia viitekehyksiä. Sen sijaan sovelluksessa hyödynnetään eri olio-ohjelmoinnin suunnittelumalleja aina tarpeen mukaan ja MVC-arkkitehtuuria.

JavaScript-kirjastona käytetään jQuerya ja sen tarjoamia lukemattomia lisäosia, joita ovat esimerkiksi käyttöliittymän komponenttien tekemistä helpottava kirjasto jQueryUI ja taulukkotietojen järjestämiseen käytettävä tablesorter-lisäosaa.

jQueryUI on sarja käyttöliittymän vuorovaikutusta, efektejä, pienoisohjelmia ja teemoja, jotka on rakennettu jQuery JavaScript -kirjaston päälle. jQueryUI:n avulla voi tehdä korkeasti interaktiivisia web-sovelluksia tai lisätä vaikka päivämäärän valinnan pienoisohjelman lomakkeelle. [7.]

Tablesorter on jQueryn lisäosa, joka on tarkoitettu taulukoiden näyttämiseen. Tablesorterin avulla taulukot voidaan järjestää ilman sivun uudelleenlataamista. Tablesorter voi jäsentää ja järjestää montaa erityyppistä dataa. [8.]

jQueryn etuna ovat aikaisempi kokemus kirjaston käytöstä Geometrix Oy:ssä ja laaja dokumentointi ja yhteisö. jQueryn kanssa käytetään rakennetta tuovaa ja reititystä helpottavaa Crossroads-JavaScript-kirjastoa. Crossroads mahdollistaa selaimen historian toimimisen Ajax-sovelluksessa.

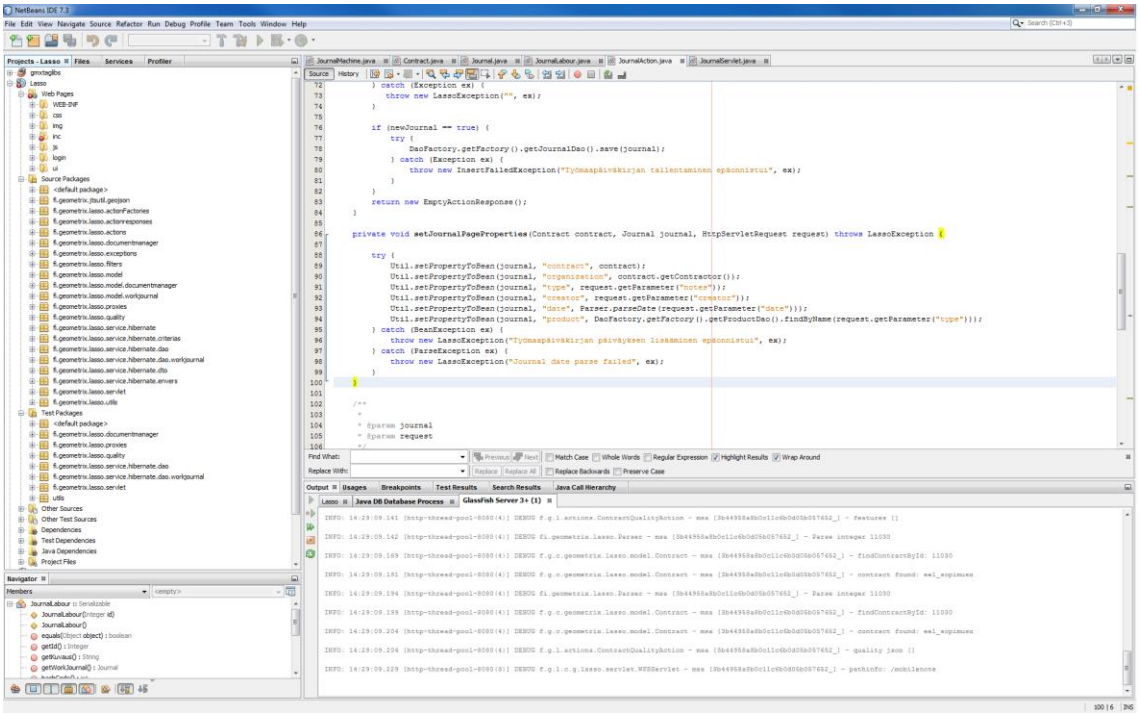
Karttakomponentti toteutetaan avoimen lähdekoodin OpenLayers-JavaScript-kirjastolla. OpenLayers on JavaScript-kirjasto, joka mahdollistaa dynaamisen kartan näyttämisen moderneilla selaimilla [15]. Sovelluksen yhtenä kartta-aineistona käytetään avointa OpenStreetMap-kartta-aineistoa, joka ladataan Geometrix Oy:n palvelimelta. OpenStreetMap on avoin ja kaikkien vapaasti muokattavissa oleva maailmakartta. Kartat ovat vapaasti ladattavissa ja käytettävissä lisenssin mukaisesti [16].

4 Kehitysympäristö

NetBeans

Sovelluksen kehitykseen tarvitaan monia työkaluja ja sovelluksia, jotta kehitys onnistuisi mahdollisimman helposti ja nopeasti. Ohjelmointiympäristönä käytetään NetBeansia. NetBeans on Oraclen kehittämä ohjelmisto, joka helpottaa ja nopeuttaa Java-sovellusten kehitystä. NetBeans on enemmän kuin tekstieditori. NetBeans kääntää koodin automaattisesti, antaa ohjelmointivinkkejä ja kertoo ohjelmointivirheistä. NetBeansissa on lukuisia pikanäppäimiä, jotka esimerkiksi sisentävät koodin automaattisesti tai näyttävät käytettävät Java-luokat, metodit tai API-kuvauksen.

API-kuvaus on osa ohjelmiston dokumentointia. Javassa lähdekoodi dokumentoidaan JavaDoc-työkalulla. Dokumentointi on kuin kommenttien kirjoittamista, mutta erityisillä avainsanoilla. NetBeans osaa tehdä lähdekoodidokumentista automaattisesti HTML-sivun [24.].



Kuva 3. Ohjelmointiympäristö.

Kuvassa 3 nähdään NetBeans IDE:n rakenne. Vasemmalla on tiedostohierarkia, jolla voidaan helposti navigoida projektin tiedostorakenteessa. Oikealla näkyy tekstieditori,

johon ohjelmakoodi kirjoitetaan. Ohjelmakoodi on värikoodattu, jotta siitä erottaa helpommin ohjelmakoodin tietyt osat. Alhaalla keskellä nähdään konsolit, joita voivat olla esimerkiksi Glassfishin-konsoli. Konsolin teksti voi olla ohjelmoijan itse kirjoittamia virheenkorjauslokeja tai Java-sovelluksen virheitä.

NetBeansissa on lukuisia toimintoja, jotka auttavat ohjelmoijaa monissa eri tapauksissa. Moduulit voivat generoida koodia automaattisesti tai vaikka ladata riippuvuuksia. Yksi yleinen moduuli on Maven.

Maven

Maven on Apachen kehittämä Java-sovellusten rakentamiseen ja ylläpitämiseen helpottava työkalu. Maven helpottaa ylläpitämään sovelluksen ulkoisia kirjastoja eli riippuvuuksia. Maven-projektiin liittyy POM-tiedosto, joka kertoo projektista tietoja ja hallitsee sen rakennetta. POM:iin kirjoitetaan esimerkiksi projektin perustiedot, riippuvuudet, kehittäjät ja resurssit[10].

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Koodiesimerkki 1. POM-tiedoston rakenne [11].

Palvelin

Jotta sovelluksen toimivuutta voidaan kokeilla, tarvitaan palvelin, jossa sovellus ajetaan, sekä tietokanta johon sovelluksen tiedot tallennetaan. NetBeansissa on mahdollista pystyttää omalle koneelle väliaikainen GlassFish-virtuaalipalvelin, johon sovellus asennetaan. Sovellus muodostaa tietokantayhteyden Geometrix Oy:n sisäisen virtuaalipalvelimen tietokantaan JDBC-resurssin kautta.

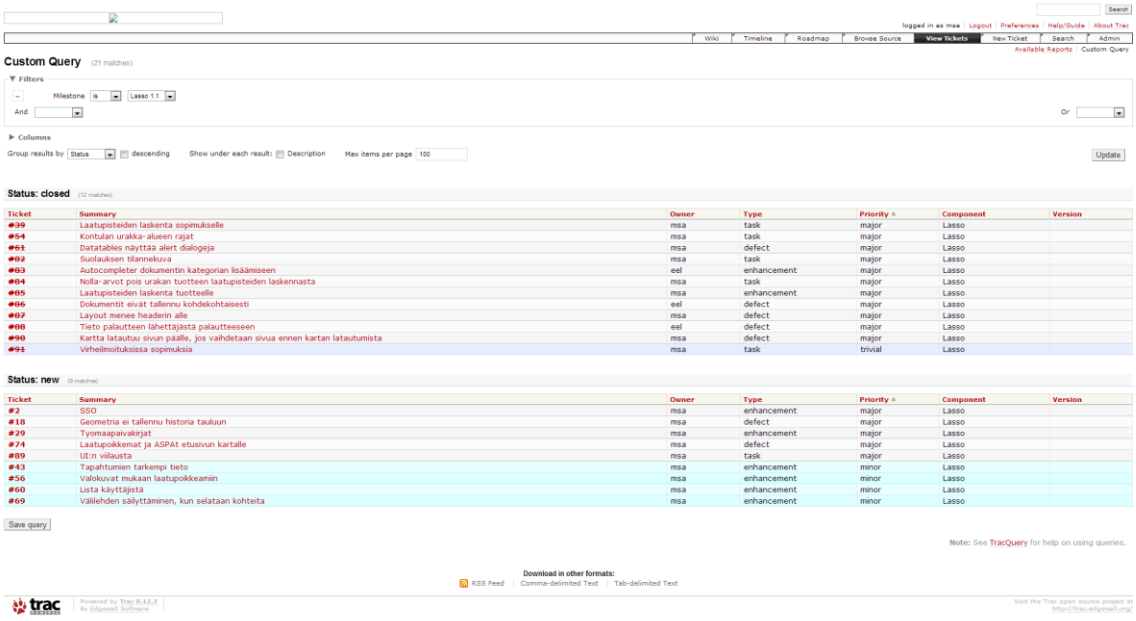
JDBC mahdollistaa SQL-lauseiden suorittamisen tietokantaan Java-koodissa [12]. JDBC-resurssin käyttöön tarvitaan yhteys. Yhteys muodostetaan JNDI-rajapinnan kaut-

ta [13]. JNDI on rajapinta, joka on määritelty Java-teknologiassa. Se mahdollistaa nimeämis- ja tiedostorakennetoiminnallisuuden Java-ohjelmointikielessä [14].

Versionhallinta

Versionhallinta on ohjelma, jolla voidaan pitää kirjaa tiedostoihin tehdyistä muutoksista. Se mahdollistaa monen ohjelmoijan ohjelmoimaan samaa koodia samanaikaisesti. Kun halutut muutokset tiedostoihin on tehty, muutokset tallennetaan versionhallintaan ja kirjoitetaan kuvaus tehdyistä muutoksista. Ohjelmoijat voivat verrata versioiden muutoksia keskenään tai palauttaa aikaisemman version. Yleisiä versionhallintaohjelmia ovat Git ja SVN. Tässä sovelluksessa käytetään SVN-versionhallintaohjelmaa.

SVN:ää voidaan käyttää TortoiseSVN-ohjelmalla. TortoiseSVN on graafinen työkalu SVN-versionhallintaohjelman käyttämiseen. TortoiseSVN:llä on mahdollista vertailla tiedostoihin tehtyjä muutoksia ja palauttaa aikaisemmat versiot. Jos muokatut tiedostot tallennetaan SVN:ään, SVN etsii, onko tiedostoon tehty muutoksia. Jos SVN:stä löytyy samaan tiedostoon tehtyjä muutoksia, SVN yrittää automaattisesti sulauttaa muutokset yhteen. Joskus automaattinen sulautus epäonnistuu ja tiedostot joudutaan sulauttamaan käsin esimerkiksi TortoiseSVN ohjelmalla. SVN:ssä on mahdollista tehdä ohjelmistokehityshaara, jonka tiedostomuutokset eivät mene päähaaraan eli trunkiin. Haara on mahdollista myöhemmin sulauttaa toiseen haaraan.



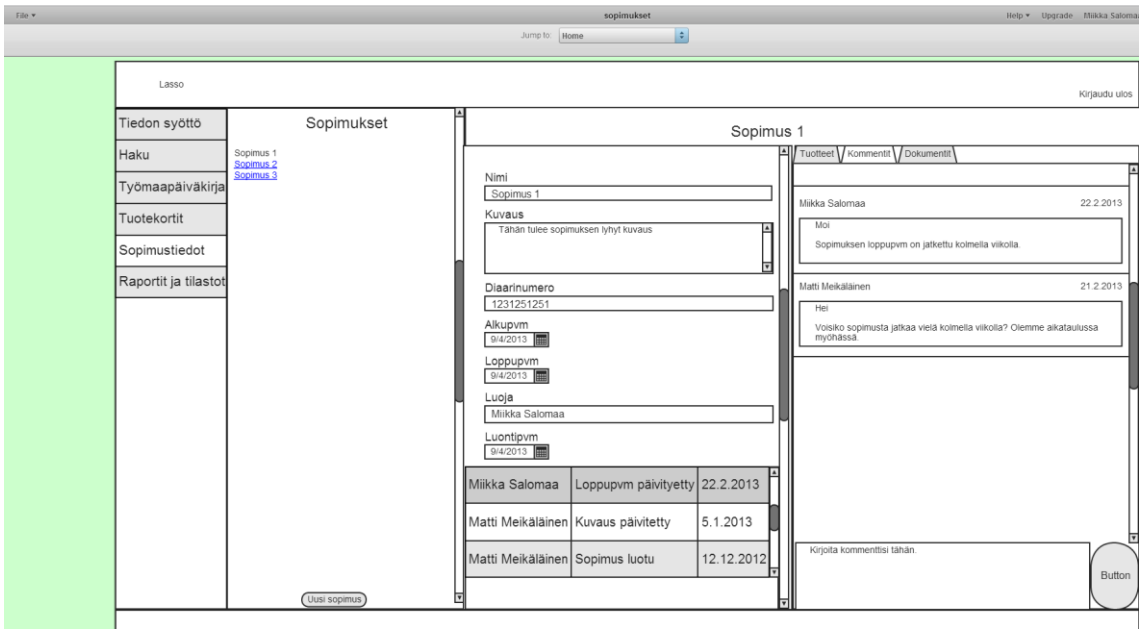
Kuva 4. Trac.

Kuvassa 4 nähdään kehityksessä käytettävän dokumentointijärjestelmän web-käyttöliittymä. Trac on dokumentointiin ja projektin hallinnointiin kehitetty työkalu. Tracissa on mahdollista kirjoittaa wiki-dokumentteja, tehdä tikettejä ja etappeja sovelluksen versioille. Tiketit ovat ohjelmoijalle määritettyjä tehtäviä, jotka ohjelmoija suorittaa. Kuvassa 4 nähdään tikettikyselyn tuloksia. Ylhäällä nähdään suljettuja tikettejä ja alhaalla uusia. Tikettiä voidaan referoida, tai se voidaan sulkea tiedoston tallentamisessa SVN:ään. Suljetut tiketit näkyvät yliviivattuina kuvassa 4. Trac lähettää ohjelmoijalle automaattisesti sähköpostia, jos häneen liittyvään tikettiin tehdään muutoksia. Näin kaikki tiketttiin liittyvät henkilöt pysyvät kartalla tehtävän tilanteesta.

5 Käyttöliittymän suunnittelu

5.1 Käyttöliittymä

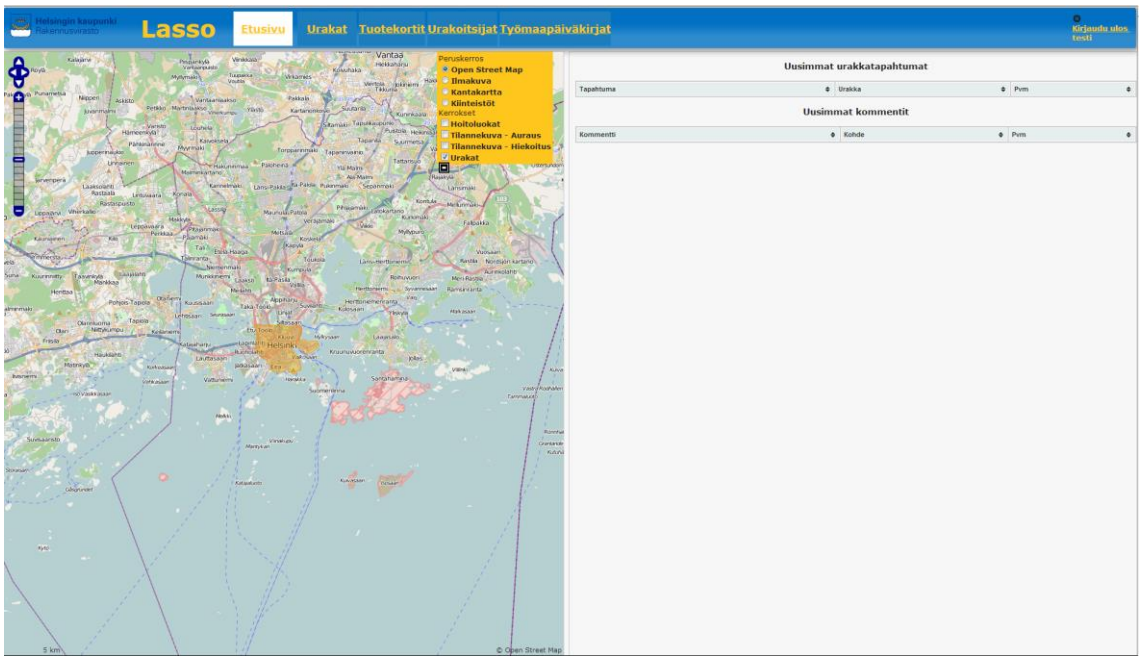
Käyttöliittymä on koko sovelluksen kehityksen ajan elävä komponentti. Käyttöliittymä muuttuu moneen kertaan sovelluksen kehityksen aikana. Sen toimivuutta on hankala arvioida ilman palautetta, siksi eri käyttöliittymäversioiden ja muutoksien kokeilu on tärkeää. Tavoitteena on lopulta löytää käytettävä, yksinkertainen, helposti opittava ja toimiva käyttöliittymä. Käyttöliittymän suunnittelu alkaa hahmottelusta.



Kuva 5. Käyttöliittymän hahmottelua Mockingbird-ohjelmalla.

Sovelluksen käyttöliittymää hahmoteltiin aluksi Mockingbird-ohjelmalla. Mockingbird on web-työkalu, joka tekee helpoksi käyttöliittymän luomisen internet-sivulle tai sovellukselle [25]. Kuvassa 5 näkyy alkuperäinen hahmotelma käyttöliittymästä.

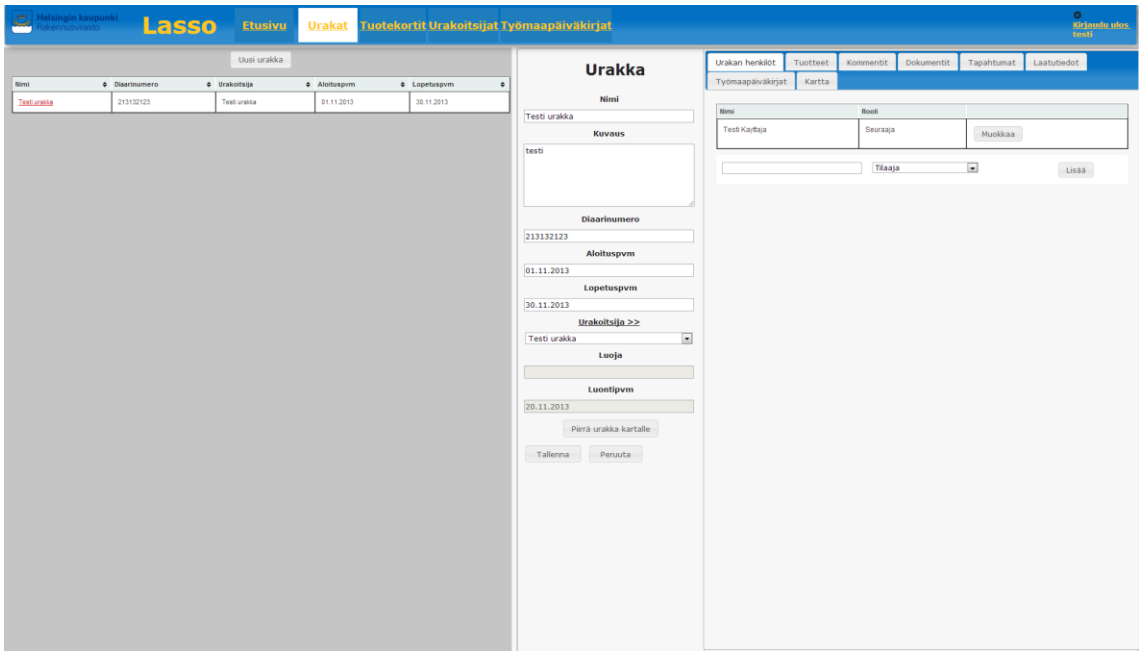
Liitteestä 1 huomataan, että käyttötapauksesta selviää tärkeimmät toiminnot ja niitä voidaan hyödyntää käyttöliittymän suunnittelussa. Liitteessä 1 nähdään esimerkiksi tietolomakkeelle ja välilehdille tulevat tiedot.



Kuva 6. Sovelluksen etusivu.

Kuvassa 6 nähdään sovelluksen etusivukäyttöliittymä. Hahmottelussa navigointi oli vertikaalisesti vasemmalla. Myöhemmin huomattiin toimivammaksi asettaa navigointi yläpalkkiin, että se ei veisi tilaa sovelluksen tiedoilta. Kuvassa 6 vasemmalla nähdään kartta, joka on toteutettu OpenLayers JavaScript-kirjastolla. Kartalle voidaan valita erilaisia karttakerroksia, joista näkyy visuaalista tietoa käyttäjälle. Oikealta käyttäjä näkee häntä kiinnostavat sovelluksen viimeiset tapahtumat ja lisätyt kommentit. Yläpalkin oikeassa nurkassa näkyy sovelluksen käyttäjän käyttäjänimi ja kirjaudu ulos painike.

Yksi käyttöliittymän tärkeimmistä toiminnollisuuksista on vasemman ja oikean osion koon muuttaminen. Kokoa voidaan muuttaa raahaamalla kuvassa 6 keskellä olevaa välikköä. Näin sovelluksen vasenta tai oikeaa puolta voidaan kasvattaa ja toista puolta pienentää.



Kuva 7. Sovelluksen Urakat käyttöliittymä.

Kuvassa 7 nähdään sovelluksen yksi keskeisimmistä käyttöliittymän osioista. Vasemmalla nähdään lista käyttäjän urakoista. Urakkalista on Tablesorter JavaScript-kirjaston ansoista mahdollista järjestää ilman sivun lataamista. Listan yläpuolelta voidaan luoda uusi urakka. Keskellä näkyy urakan tiedot, joita voidaan muokata helposti. ”Piirrä urakka kartalle” – painikkeesta avautuvat kartta-välilehti ja urakan rajojen piirtotyökalut. Oikealla on erilaisia tietoja ja toimintoja välilehdillä. Toimintoja ovat esimerkiksi kommentit, dokumenttien hallinta ja kartta.

5.2 Käytettävyys

Käytettävyyteen on syytä erityisesti panostaa, jos sovelluksen käyttämistä ei erikseen opeteta. Tätä sovellusta käyttävät monet eri henkilöt eri taustoista ja heidän pitää pystyä tekemään töitä sovelluksen parissa ilman ongelmia. Käytettävyyteen vaikuttavia tekijöitä on monia. Web-kehityksessä on kahdeksan yleistä ongelmaa, jotka eivät ole muuttuneet vuosien saatossa.

- Linkit eivät vaihda väriä vierailun jälkeen.
- Sivu avaa uusia selaimen ikkunoita.
- Selaimen takaisinpainike ei toimi sivulla.

- Sivu avaa ponnahdusikkunoita.
- Suunnitteluelementit näyttävät mainoksilta.
- Sivu rikkoo webin yleissopimuksia.
- Sivulla on usvaista sisältöä ja tyhjää hypetystä.
- Sivulla on tiheää sisältöä ja vaikeasti luettavaa tekstiä [33, s.60.].

Sovelluksen käyttöliittymän suunnittelussa käytiin läpi muutamia käytettävyysongelmia. Esimerkiksi selaimen takaisinpainiketta ei haluttu rikkoa. Selaimen takaisipainike antaa turvan käyttäjälle tutkia sivua tietäen aina toiminnon josta pääsee takaisin edelliseen tilanteeseen. Takaisin-painike on yleensä selaimen toiseksi käytetyin toiminto. Takaisin-painike on aina turvallisesti samassa paikassa. Jotkin sovellukset hyödyntävät omia ”peruuta”-painikkeita, mutta niiden käyttö on vaikea oppia. Käyttäjät ovat tottuneet selaimen takaisin-painikkeeseen ja siihen että sillä voidaan aina kumoa tehty toiminto. [33, s.63–65.].

Selaimen takaisinpainikkeen toiminta edellyttää URL-osoitteen muuttumista. Sovellus päätettiin tehdä yhden sivun sovellukseksi, eli sovellus ei tee sivun vaihtoja, vaan sisältöä päivitetään palvelimelta Ajax-tekniikoilla. Tämä kuitenkin rikkoo takaisin-painikkeen toiminnan selaimessa tavalla, jolla sen oletetaan toimivan. Ongelma korjattiin sovelluksen reitittämisellä Crossroadsjs-JavaScript-kirjaston avulla.

```
crossroads.addRoute('/urakat/', function() {
    loadContractsPage();
    showList();


});
crossroads.addRoute('/urakat/{id}', function(id) {

    loadContractsPage(id);
    loadContractPage(id);
});

crossroads.addRoute("/urakat/{id*}/{tab}", function(id, tab) {
    //TODO: Set right tab when page loads.
    loadContractsPage(id);
    loadContractPage(id, tab);
});
```

Koodiesimerkki 2 Sovelluksen reititys JavaScriptissä

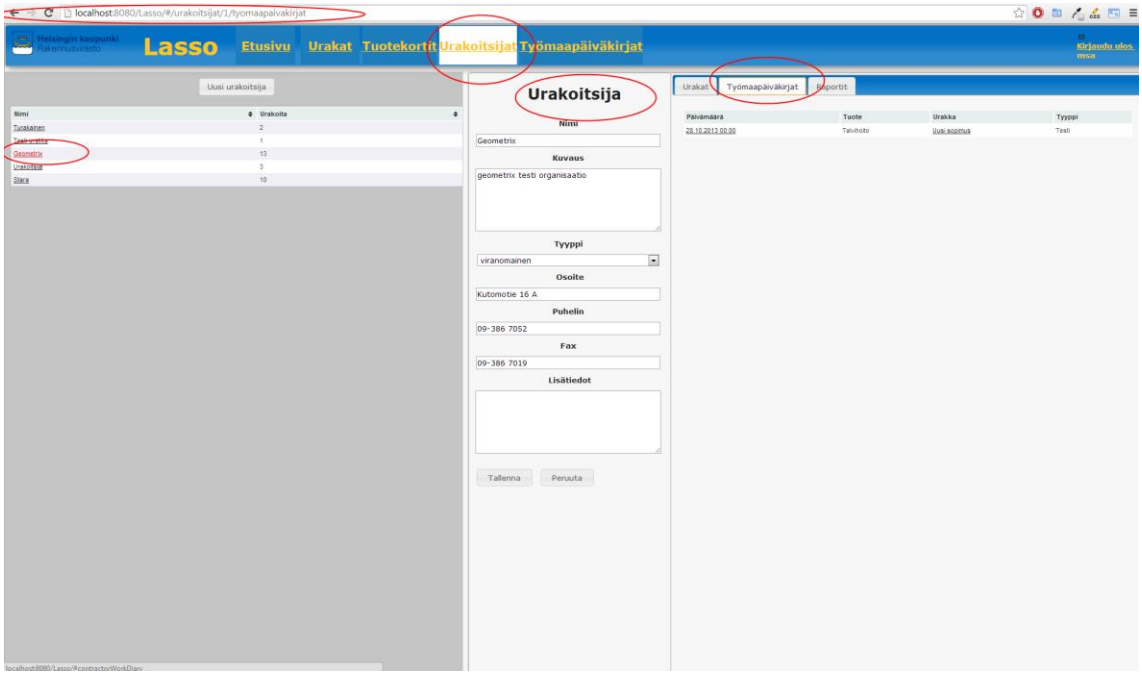
Koodiesimerkissä 2 sovelluksen reititykset tehdään Crossroadsjs-kirjastolla. Crossroads-oliolle annetaan kuunneltava osoite metodilla *addRoute* ja funktio, joka suoritetaan, jos selaimen osoite sopii yhteen parametrina annettuun merkkijonoon. Esimerkiksi selaimen osoite localhost:8080/Lasso/#urakat/10 kutsuu funktiota *loadContractsPage* ja antaa parametriksi 10. Edellinen osoite tallennetaan selaimen historiaan ja selaimen takaisin-painiketta painettaessa edellinen sivu historiasta latautuu ja selaimen osoitekenttä päivittyy.

Avattaessa dokumenttia voidaan avata se selaimen uuteen ikkunaan. Käyttäjät ovat tottuneet sulkemaan dokumentit selaimen ruksista, ja jos dokumenttia ei avata uuteen ikkunaan, saattaa käyttäjä vahingossa sulkea sovelluksen välilehden ja näin joutua taas etsimään dokumentin uudelleen [33, s.67–70].

Vaikeasti luettava teksti saattaa helposti tulla ongelmaksi, jos tietoa täytyy saada paljon pienelle alueelle. Web-sivuilla tekstin tulisi olla lyhyttä, helposti luettavaa ja helposti lähestyttävää. Tyypillisesti web-sivulle tulisi kirjoittaa puolet vähemmän tekstiä, kuin painetulle sivulle [33, s.81]. Tässä sovelluksessa tiedot on sijoitettu helposti luettaviin taulukoihin, eikä pitkiä tekstejä ole juuri missään.

Navigoinnin suunnittelussa voidaan tehdä monia virheitä. Navigointipainikkeiden tulisi olla yksinkertaisia ja aina samassa paikassa. Navigointipainikkeet on tarkoitettu navi-

gointiin, ei niinkään tyylin luomiseen. Navigoidessaan käyttäjä etsii sivulta lyhyitä sanoja, siksi navigointiin tarkoitettujen tekstien tulisi olla selkeitä ja lyhyitä [33, s.178–193.].



Kuva 8. Sovelluksessa navigointi.

Tärkein kysymys navigoinnissa on ”Missä minä olen?”. Käyttäjät eivät ymmärrä sivun rakenteesta mitään, jos he eivät tiedä missä ovat. Käyttäjät eivät tiedä missä he ovat ilman, että heille kerrotaan siitä [34, s.188–189]. Kuvassa 8 havainnollistetaan kuinka sovelluksessa kerrotaan käyttäjälle monella eri tavalla missä osiossa hän tällä hetkellä on. Selaimen osoiteriviltä voidaan päätellä, että käyttäjä on urakoitsija-osioissa ja työmaapäiväkirja-välilehdellä. Navigaatiopalkissa urakoitsijat-painike on korostettu eri värillä kuin muut navigaatiopalkissa olevat painikkeet. Vasemmalta listalta valittu urakoitsija on korostettu punaisella. Keskellä kerrotaan vielä, että tietolomake liittyy urakoitsijaan, koska sovelluksen muut tietolomakkeet ovat melkein identtisen näköisiä, kuten kuvassa 7 nähdään. Lopuksi oikealla auki oleva välilehti on korostettu eri värillä kuin muut välilehdet. Näillä käytettävyyden osioilla varmistetaan, että käyttäjä tietää missä kohtaa sovellusta hän on ja mitä hän on tekemässä.

5.3 Palvelin

Sovellus asennetaan aluksi kehitykseen ja testaamiseen luodulle Linux-virtuaalipalvelimelle Geometrix Oy:n sisäverkkoon. Kehityspalvelinympäristö on kopio tuotantopalvelimesta, johon sovellus lopuksi asennetaan. Linux-palvelimelle on asennettu valmiiksi tarvittavat ohjelmat: GlassFish-ajoympäristö, Apache-palvelin ja PostgreSQL-tietokanta.

Tietokanta on laajennettu PostgreSQL-tietokanta nimeltään PostGIS ja se mahdollistaa paikkatietoon liittyviä toimintoja tietokannassa, esimerkiksi maantieteellisten pisteiden tallentaminen tietokantaan ja sijaintiin liittyviä hakuja [17].

GlassFish on avoimen lähdekoodin sovelluspalvelin, jossa voidaan ajaa Java Enterprice -sovelluksia. Sovellus pakataan war-tiedostoon ja asennetaan GlassFishiin sovelluksena. Palvelimella on asennettuna myös Geometrix Oy:n muita web-sovelluksia. Koska tämä sovellus on täysin erillinen, sille tehdään oma toimialue (domain). Toimialueen luonti tehdään Linuxin komentoriviltä, johon luodaan etäyhteys suojatulla SSH-yhteydellä Putty-ohjelman avulla. Toimialue luontiin tarvitaan muutama Linux-komento kirjoitettuna komentoriviin:

1. Luodaan uusi toimialue (domain)*: /opt/glassfish3/bin/asadmin create-domain --adminport 5858 lasso*.

2. Vaihdetaan ylläpitäjän salasana: *glassfish3/bin/asadmin change-admin-password --domain_name lasso.*

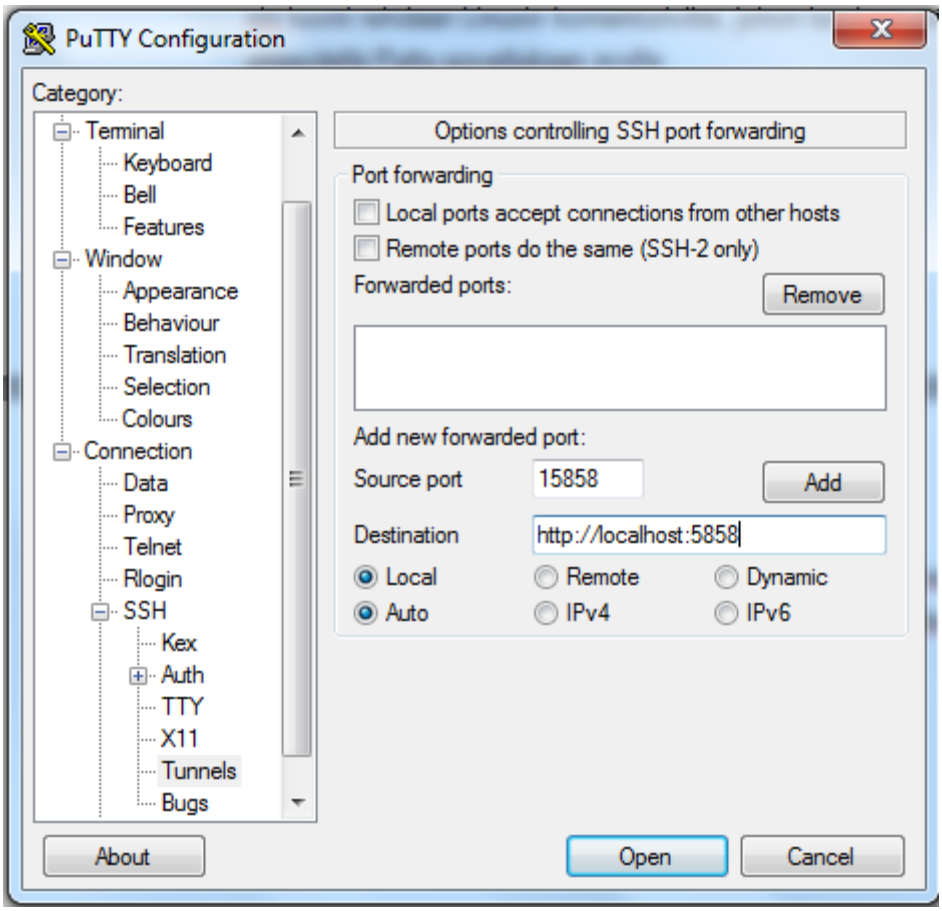
3. Luodaan toimialueesta Linux-service: *glassfish3/bin/asadmin create-service --serviceuser glassfish lasso.*

4. Käynnistetään toimialue: *glassfish3/bin/asadmin start-domain lasso.*

5. Mahdollistetaan admin-paneeliin käyttäminen: *glassfish3/bin/asadmin --port 5858 enable-secure-admin.*

GlassFishissä on myös selainpohjainen admin-hallintapaneeli. Toimialueen luonnin jälkeen admin-paneeliin on mahdollista päästä portista 5858. Tuotantopalvelimen ad-

min-hallintapaneeliin on mahdollista päästä vain SSH-tunnelin kautta, joka tehdään myös Putty-ohjelmalla.



Kuva 9. SSH-putken muodostaminen Putty-ohjelmalla.

5.4 Karttapalvelu

Karttapalvelu on Geometrix Oy:n verkossa toimiva WMS-palvelu. WMS on standardi http-rajapinta paikkatietoon liittyvien kuvien kyselyyn paikkatietotietokannasta. WMS-kysely määrittelee paikkatiedon tasot tai tason ja prosessoitavan alueen. Vastaus on yksi tai useampi paikkatietorekisteröity karttakuva, jonka selain pystyy näyttämään. [18.].

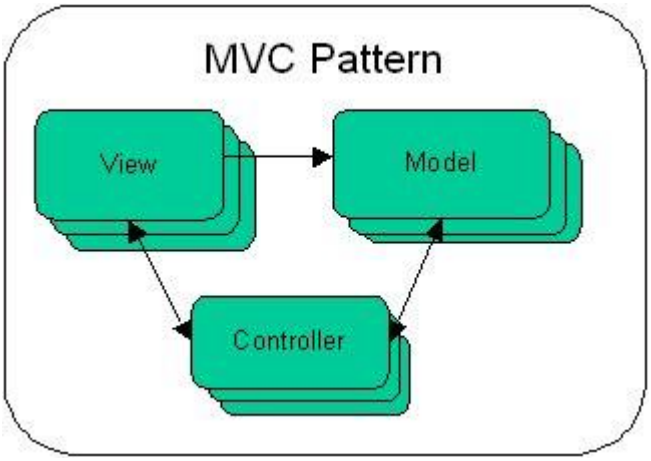
Sovelluksessa käytetään hyväksi, myös Geometrix Oy:n WFS-palvelua kohteiden hakemiseen kartalle. WFS on standardivaihtoehto FTP-protokollalle, kuinka paikkatietoa luodaan, muokataan ja vaihdetaan internetissä. Esimerkiksi WFS tarjoaa suoran pää-

syn paikkatietoon kohteen ja kohteen ominaisuuden tasolla. WFS sallii asiakkaan vastaanottaa ja muokata tietoa, jota he etsivät ilman tiedoston lähettämistä. [19.].

6 Toteutus

6.1 MVC-arkkitehtuuri

Sovellus on rakennettu MVC-arkkitehtuurin pohjalle, jossa malli, näkymä ja ohjain ovat eroteltu toisistaan riippumattomiksi komponenteiksi, kuten kuvasta 20 nähdään.



Kuva 10. MVC-arkkitehtuuri [20].

6.1.1 Malli

Malli (model) on joukko Java-entity-luokkia, jotka generoidaan tietokannasta. Entity on Javan standardi JavaBean-olio. JavaBeanin ominaisuudet ovat näkyvyydeltään *private* tai *protected.* Jokaisella ominaisuudella täytyy olla get- ja set-metodit, jos ominaisuus on *boolen*-tyyppiä sillä tulee olla is-metodi. Tyypillisesti metodit on nimetty luokkamuuttujien mukaan esimerkiksi luokkamuuttujalle *id* set- ja get-metodit ovat *public void setId(int id)* ja public int *getId().*

Entity-olio on kevyt vähän muistia vievä pysyvä toimialue olio. Tyypillisesti entity-luokka edustaa yhtä relaatiotietokannan taulua ja jokainen entity-luokan jäsenmuuttuja tieto-

kannan saraketta. Entity-luokan ominaisuuksia käytetään kartoittamaan relaatiotietokannan ja olion yhteydet. Entity-luokan täytyy täyttää seuraavat ehdot:

- Luokka täytyy annotioida *javax.persistence.Entity*-huomiolla. Huomio on Java-ohjelmointikielen metatietoa, joka kertoo tietoa ohjelmasta tai ohjelman osiosta. Huomio voi esimerkiksi antaa informaatioita ohjelmakoodin kääntäjälle [21].

- Luokalla täytyy olla argumentiton konstruktori. Luokalla voi olla muita konstruktoreita.

- Luokalla ei saa olla final-määritettä. Luokan metodit tai muuttujat eivät myöskään saa määritellä final-määritettä.

- Jos luokka annetaan eteenpäin arvona, sen tulee implementoida Serializable–rajapinta.

- Luokka voi periä sekä toisen entity-luokan tai ei-entity-luokan. Toiset ei-entity-luokat voivat myös periä entity-luokan.

- Entity-luokan luokkamuuttujat tulee olla näkyvyydeltään *private* tai *protected*, jolloin toiset luokat eivät pääse suoraan niihin käsiksi.

Tietokantataulun *primary key* -sarake voidaan esittää *javax.persistence.Id*-huomiolla. Entity-luokkien väliset relaatiot määritellään huomio, joita ovat *one-to-one*, *one-to-many*, *many-to-one*, ja *many-to-many*. Relaatiot voivat olla yksisuuntaisia tai monisuuntaisia. [22.].

6.1.2 Näkymä

Näkymä (view) on käyttöliittymä, jonka loppukäyttäjä näkee. Tässä sovelluksessa näkymä on joukko JSP-sivuja. JSP tarjoaa helpon ja nopean tavan tehdä dynaamista web-sisältöä. JSP-teknologia käyttää XML-tyylisiä merkkejä pakkaamaan logiikan, joka luo sisällön sivulle. JSP on servlet-teknologian lisäosa. Servletit ovat alustariippumattomia palvelinpuolen moduuleita, jotka sopivat saumattomasti web-palvelimen viiteke-

hykseen ja niitä voidaan käyttää web-palvelimen kykyjen kasvattamiseen pienimmällä työllä, ylläpidolla ja tuella.

JSP-teknologialla on seuraavia etuja kehittäjälle:

- JSP-teknologiaa voidaan käyttää ilman Java-ohjelmointikielen opettelemista. JSP-teknologiaa voidaan käyttää ilman Java-scriptlettejen kirjoittamista.

- Kehittäjät ja suunnittelijat voivat laajentaa JSP-kieltä yksinkertaisilla ”merkkikäsittelijöillä”.

- JSP on integroitu JSTL-lausekekieleen ja päivitetty tukemaan funktioita. [23.].

JSP-sivuille kirjoitetaan HTML-ohjelmointikieltä, jolla saadaan tehtyä käyttöliittymäkomponentteja. JSP-sivun tyylit tehdään CSS-tyylitiedostoilla ja interaktiivisuutta sivulle saadaan jQuery- ja jQueryUI-JavaScript-kirjastoilla.

6.1.3 Ohjain

Ohjain (controller) eli sovelluksen logiikka on vastuussa tiedon hakemisesta tietokannasta entity-olioihin ja niiden lähettämisen JSP-sivulle. Ohjain on Java Servlet -luokka, jota kutsutaan, kun tiettyyn osoitteeseen tehdään http-pyyntö. Osoitteet on määritelty projektin *web.xml*-tiedostossa.

Servletit ovat Java-alustan teknologioita, joilla on mahdollisuus kasvattaa ja tehostaa web-palvelimia. Servletit tarjoavat komponenttipohjaisen, alustariippumattoman metodin web-pohjaisten sovellusten rakentamiseen ilman CGI-ohjelmien suorituskykyrajoitteita. Servleiteillä on pääsy koko Java API - rajapintaan, mukaan lukien JDBC API:in. Servleteillä on myös pääsy http-spesifisiin kutsuihin [27].

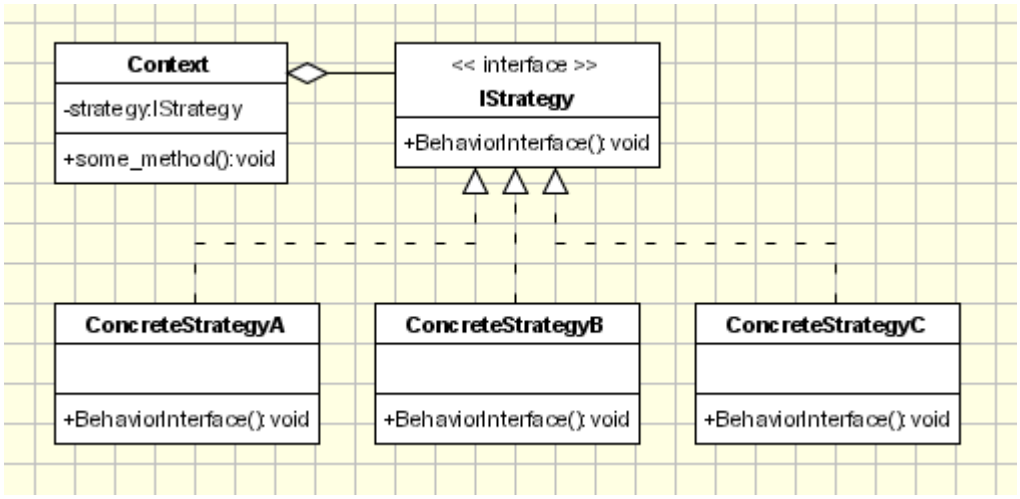
Ohjain toimii rajapintana tietomallin ja käyttöliittymän välillä. Ohjain hakee tietokannasta entity-luokkia hyödyntäen Hibernatea ja muita luokkia.

Hibernaten tehtävänä on käsitellä Java-olioiden relaatioita. Hibernate on ilmainen ja avoimen lähdekoodin Java-paketti, joka tekee relaatiotietokantojen käsittelyn helpoiksi.

Hibernate synkronoi tietokannan entity-luokkien kanssa ja näin helpottaa tietokannan ylläpitämistä Java-ohjelmalla. Se mahdollistaa keskittymisen sovelluksen olioihin ja toimintoihin ilman, että niiden tallentamisesta tai löytämisestä myöhemmin täytyy huolehtia. [26.].

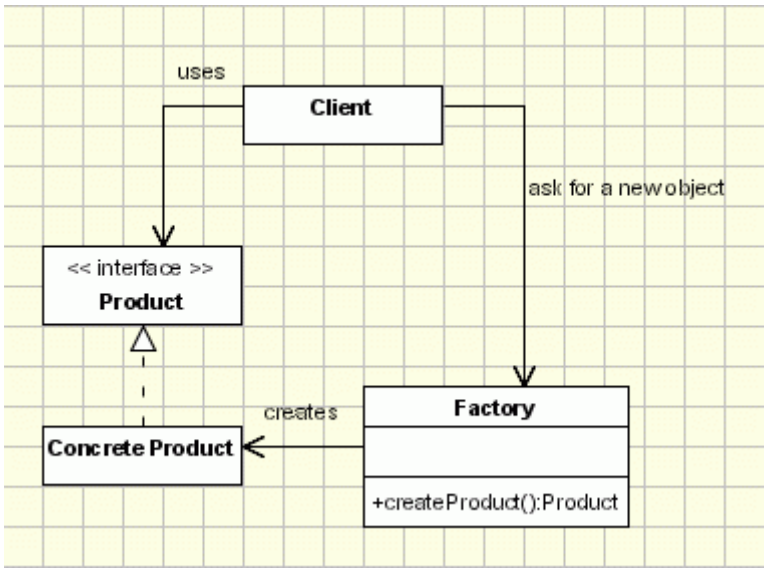
6.2 Tapahtumien kulku

Sovelluksessa hyödynnetään olio-ohjelmoinnin suunnittelumalleja, jotka tekevät ohjelmakoodista helposti luettavan, testattavan, uudelleenkäytettävän ja laajentuvan. Tietojen haut ja tallennukset noudattavat tiettyä kaavaa aina käyttöliittymästä tietokantahakuun asti. Sovelluksessa käytettäviä tunnettuja suunnittelumalleja ovat esimerkiksi *Strategy Pattern ja Factor pattern.*



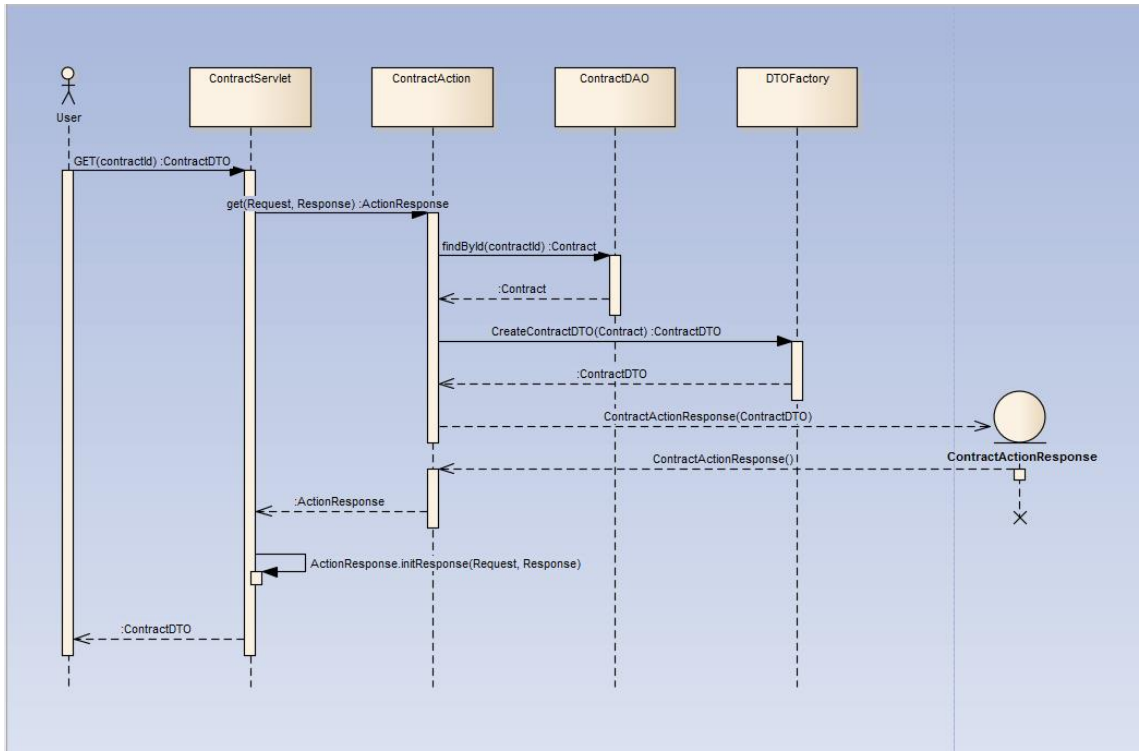
Kuva 11. Strategy pattern [28].

Kuvassa 11 nähdään, että Strategy määrittää yleisen rajapinnan kaikille sovelluksessa käytettäville algoritmeille. Context käyttää tätä rajapintaa kutsuessa ConcreteStrategyssä määriteltyä algoritmia. [.28].



Kuva 12. Factory pattern [29]

Factory method- patternissa Factory luo olioita ilman logiikan paljastamista kutsujalle. Kuvassa 12 nähtävä Client pyytää Factorylta Product-oliota. Factory luo Product-rajapinnan toteuttavan olion ja palauttaa sen takaisin Clientille.



Kuva 13. Sopimuksen haun tapahtumien kulku

Sovelluksessa Strategy-rajapintana toimii Action-rajapinta, jolla on metodit get, post ja delete. Metodit palauttavat ActionResponse-rajapinnan, jolla on metodi initResponse.

Metodi on algoritmi, joka on vastuussa siitä kuinka tieto palautetaan kutsujalle. Sovelluksessa on myös monta erilaista Factory pattern -suunnittelumallin mukaista luokkaa. Yksi näistä on DTOFactory-luokka, joka on vastuussa DTO-olioiden luomisesta.

Data Transfer Object -olioon on kapseloitu JSP-sivulle välitettävät ja vain välttämättömät tiedot. Kapseloimisella tarkoitetaan, että tiedot ovat vain luettavissa, eikä niitä voi asettaa kuin konstruktorissa oliota luodessa. DTO-oliot vähentävät tietoliikennekuormaa. Suuri tietoliikennedatamäärä saattaa hidastaa sovelluksen toimintaa varsinkin hitailla tietoliikenneyhteyksillä. Toinen syy DTO-olioiden käyttämiseen on turhien riippuvuuksien poistaminen. Sovelluksen ei tarvitse säilyttää muistissa käyttämättömiä olioita turhaan.

Kuvasta 13 nähdään, mitä tapahtuu, kun käyttäjä pyytää yhden sopimuksen tietoja sovelluksesta. Aluksi sovellus tekee http-pyynnön tiettyyn osoitteeseen. Tähän osoitteeseen kartoitettu Servlet käsittelee pyynnön. Pyyntö menee Servletin doGet-metodille, joka luo ContractAction Action-rajapinnan toteuttavan olion ja kutsuu tämän get-metodia ja antaa sille parametrina HttpServletRequest- ja HttpServletResponse-oliot, jotka muodostuvat http-pyynnöstä. ContractAction luo DAO–rajapinnan toteuttava ContractDAO-olion. Data Access Object on rajapinta tietokantaan. Esimerkiksi ContractDao on vastuussa Contract-olioiden hakemisesta tietokannasta Hibernaten Criteria-olioiden avulla.

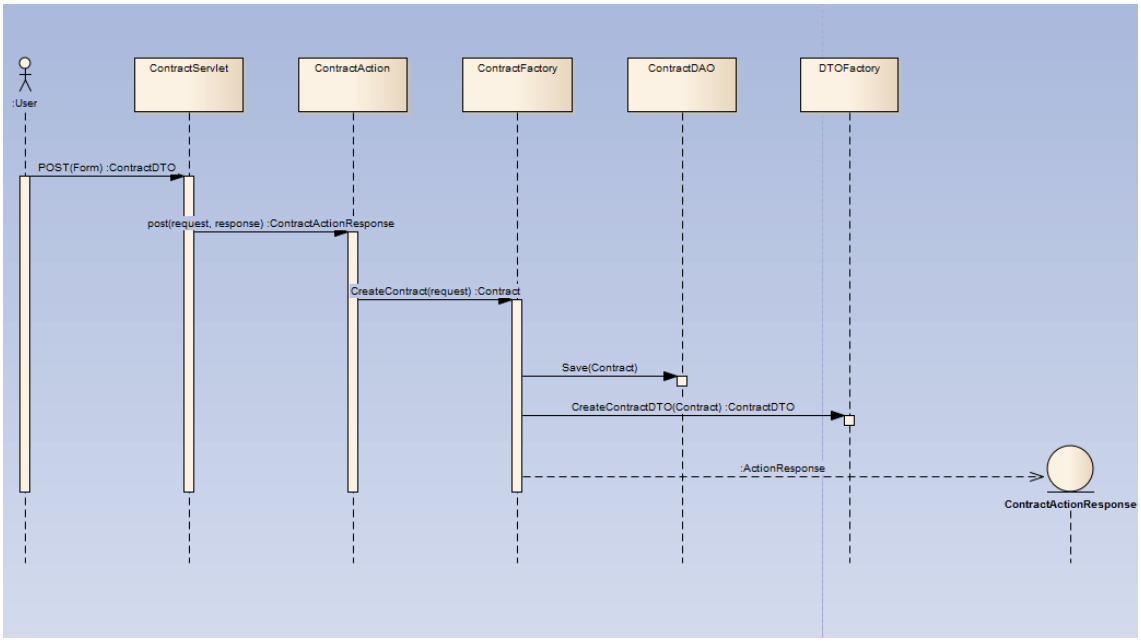
```
public Contract findById(int id){
    Criteria criteria = HibernateUtil.GetSession().createCriteria(Contract.class);
    criteria.addRestriction(Restrictions.eq("id",id));
    List<Contract> contracts = criteria.list();
    if(!contracts.isEmpty){
        return contracts.get(0);
    }
    return null;
}
```

Koodiesimerkki 3. Contract-olion hakeminen tietokannasta.

Koodiesimerkissä 3 nähdään, kuinka Hibernaten Criteria-olioiden avulla haetaan Contract-olio tietokannasta. Aluksi Criteria-olio luodaan Hibernaten Sessio-olioista. Parametrina sille annetaan haettava Entity-luokka. Criteria-oliolle annetaan rajoitus, jonka mukaisesti tietokannasta haetaan riviä, jonka id on metodin parametrina tullut kokonaisluku. Lopuksi Criteria-olion löytämät Contract-oliot asetetaan Contracts-muuttujaan

ja palautetaan sen ensimmäinen alkio. Listassa on varmati joko yksi alkio tai ei yhtään, koska id on taulun pääavain ja se on aina uniikki.

Kuvassa 13 lopuksi ContractDao-oliolta saatu Contract-olio annetaan parametrina DTOFactorylle, joka muodostaa siitä ContractDTO-olion. ContractDTO-olio annetaan parametrina ContractActionResponse-oliolle joka palautetaan Servletille. Servlet kutsuu sen initResponse-metodia, joka tässä tapauksessa lähettää ContractDTO-olion JSP-sivulle.



Kuva 14. Sopimuksen tallennus.

Kuvassa 14 kuvataan sopimuksen tallentamisen tapahtumien kulkua. Tapahtumien kulku alkaa siitä, kun käyttäjä tallentaa sopimuksen tietolomakkeen JSP-sivulta. Tallennus tekee http-post-pyynnön ContractServletille. Servletistä pyyntö menee ContractAction-oliolle, joka pyytää ContractFactoryn muodostamaan post-parametreista Contract-olion. ContractDao-olio tallentaa sen tietokantaan Hibernatea hyödyntäen.

6.3 Karttakomponentti

Sovelluksessa paikkatieto on tärkeässä roolissa. Paikkatieto helpottaa hahmottamaan loppukäyttäjille monia komponentteja ja näyttämään dataa visuaalisessa muodossa. Jokaisen sopimuksen paikkatieto perustuu multipolygon-geometriaan, jota käytetään sopimuksen maantieteellisten rajojen hahmottamisessa ja WFS-kyselyiden suodattamisessa. Multipolygon on instanssi monia polygon-instansseja, jotka ovat geometriassa

monikulmioita. Loppukäyttäjällä on mahdollisuus piirtää kartalle sopimuksen rajat OpenLayers JavaScript-kirjaston piirtotyökaluilla. Kun sopimuksen rajat ovat piirretty ja sopimus tallennetaan, myös sopimuksen geometria tallennetaan PostGIS-tietokantaan. Sopimukset näkyvät kartalla etusivulla ja sopimuksen karttakomponentilla.

Sopimuksen maantieteellisten rajojen alueelta sovellus hakee Geometrix Oy:n Mobilenote-mobiilisovelluksella tehtyjä merkintöjä kartalle WFS-rajapinnan avulla. Sovellus hyödyntää Geometrixin Java-kirjastoa, joka toimii rajapintana Mobilenote WFS -palvelun ja kartan välillä. Kirjaston avulla WFS-kyselyyn voidaan asettaa suodattimia, kuten sopimuksen geometria tai haettavat kohteet. Kohteita voivat olla esimerkiksi laatupoikkeamamerkinnät.

Kartalla näytetään monia karttatasoja. Karttatasot tulevat Geometrix Oy:n WMS-rajapinnan kautta. WMS- ja WFS -pyynnöt eivät suoraan mene JavaScriptistä Geometrixin rajapintaan, vaan välityspalvelin-servletin kautta, joka tekee todennuksen, jotta rajapintaa voidaan käyttää.

```
var mapProperties = MapProperties();
mapProperties.mapBaseLayers.forEach(function(layer) {
    map.addLayer(new OpenLayers.Layer.WMS(
            layer.layerDescription,
            "map",
            {
                layers: layer.layerName,
                format: "image/png"
            },
    {
        buffer: 0,
        attribution: layer.copyright,
        transitionEffect: 'resize'
    }
    ));
});
```

Koodiesimerkki 4. WMS-tason lisääminen OpenLayers-kartalle.

Koodiesimerkissä 4 nähdään, kuinka OpenLayers JavaScript-kirjaston avulla lisätään kartalle WMS-taso. Kartan perustasot on asetettu *MapProperties*-olion *mapBaseLayers*-ryhmään. Ryhmä käydään alkio kerrallaan läpi ja asetetaan uusi OpenLayers WMS-taso kartalle. Tasolle annetaan parametrina esimerkiksi kuvaus ja HTML-

elementin id, jossa kartta sijaitsee, tason nimi ja formaatti. Tämän jälkeen kartta osaa muodostaa oikeanlaisen kyselyn karttaruutujen hakemiseen palvelimelta.

```
http://localhost:8080/Lasso/map?LAYERS=osm2&FORMAT=image
%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&STYLES=&SRS=EPSG
%3A2393&BBOX=3401408,6680224,3409600,6688416&WIDTH=256&HEIGHT=256
```

Koodiesimerkki 5 WMS-kysely.

Koodiesimerkissä 5 nähdään esimerkki http-kyselystä, joka lähetetään palvelimelle. Kyselyssä on parametreina tietoja, jolla karttapalvelin osaa palauttaa oikeanlaisen karttaruudun. Koodiesimerkissä 5 parametreja ovat osm2-taso, image/png-formaatti, wms-palvelu, versio 1.1.1, GetMap-kysely, EPSG2392-koordinaatisto, alue ja koko.

Kuva 15. Karttaruutu.

Kuvassa 15 nähdään koodiesimerkin kyselyn vastaus. Vastauksen formaatti on määritelty kyselyn parametrissa ja tässä tapauksessa vastaus on png-kuva.

```
laatupoikkeamaLayer = new OpenLayers.Layer.Vector(
        "Laatupoikkeamat",
        {
            styleMap: new OpenLayers.StyleMap({
                "default": new OpenLayers.Style({
                    strokeColor: "#000",
                    strokeOpacity: .7,
                    strokeWidth: 1,
                    fillColor: "#ff0000",
                    pointRadius: "6"
                }),
                "temporary": new OpenLayers.Style({
                    strokeColor: "#0033ff",
                    strokeWidth: 2,
                    fillColor: "#0033ff",
                    pointRadius: "6"

                }),
                "select": new OpenLayers.Style({
                    strokeColor: "#000",
                    strokeOpacity: .9,
                    strokeWidth: 2,
                    fillColor: "#0033ff",
                    pointRadius: "6"
                })
            }),
            strategies: [new OpenLayers.Strategy.BBOX({resFactor: 1}), filterStrategy],

            protocol: new OpenLayers.Protocol.WFS({
                srsName: "EPSG:2393",
                version: "1.1.0",
                url: "wfs/mobilenote",
                featureType: "laatupoikkeama", //geoserver Layer Name,
                featurePrefix: "note",
                featureNS: "http://www.geometrix.fi/mobilenote", // Edit Workspace Namespace URI
            })
        }

);
```

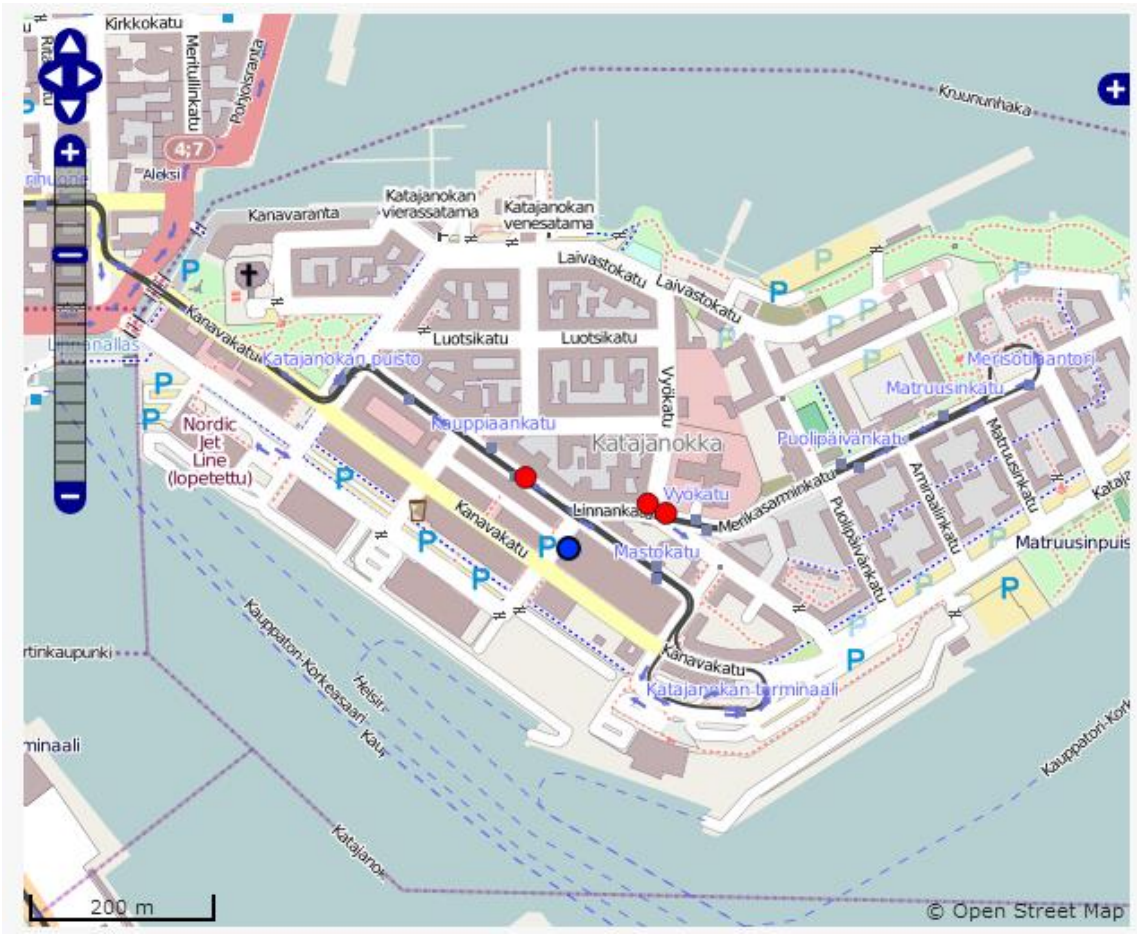
Koodiesimerkki. 6 WFS-taso.

Koodiesimerkissä 6 muodostetaan WFS-kysely OpenLayers JavaScript-kirjaston avulla. Aluksi palautettaville kohteille määritellään tyylit. Ensimmäisenä määritellään perustyyli, toisena väliaikainen tyyli ja kolmantena tyyli, jota käytetään kun kohde on valittuna kartalta. Strategies-kohdassa määritellään, milloin kohteet päivitetään. Tässä kohteet päivitetään aina, kun kartan taso muuttuu. Lopuksi määritellään tason protokollaksi WFS. *OpenLayers.Protocol.WFS*-oliolle annetaan parametrina koordinaatisto, versio, osoite ja kyseltävän kohteen tiedot.

```
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs" service="WFS" version="1.1.0" xsi:schemaLocation="http://www.opengis.net/wfs http://schemas.opengis.net/wfs/1.1.0/wfs.xsd"
    <wfs:Query typeName="note:laatupoikkeama" srsName="EPSG:2393" xmlns:note="http://www.geometrix.fi/mobilenote">
        <ogc:Filter xmlns:ogc="http://www.opengis.net/ogc">
            <ogc:And>
                <ogc:Intersects>
                    <ogc:PropertyName>geom</ogc:PropertyName>
                    <gml:MultiSurface xmlns:gml="http://www.opengis.net/gml" srsName="EPSG:2393">
                        <gml:surfaceMember>
                            <gml:Polygon>
                                <gml:exterior>
                                    <gml:LinearRing>
                                        <gml:posList>3383892 6676014.5 3383892 6671934.5 3389796 6672238.5 3388996 6675998.5 3383892 6676014.5</gml:posList>
                                    </gml:LinearRing>
                                </gml:exterior>
                            </gml:Polygon>
                        </gml:surfaceMember>
                    </gml:MultiSurface>
                </ogc:Intersects>
                <ogc:BBOX>
                    <ogc:PropertyName>geom</ogc:PropertyName>
                    <gml:Envelope xmlns:gml="http://www.opengis.net/gml" srsName="EPSG:2393">
                        <gml:lowerCorner>3384309.6790321 6672411.5</gml:lowerCorner>
                        <gml:upperCorner>3389253.6790321 6676483.5</gml:upperCorner>
                    </gml:Envelope>
                </ogc:BBOX>
            </ogc:And>
        </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

Koodiesimerkki 7. WFS-kysely.

Koodiesimerkissä 7 on esimerkki WFS-kyselystä, joka tehdään palvelimelle. WFS-kysely käyttää http-post-metodia, joka sisältää kyselyn XML-merkkikielellä. WFS-palvelin palauttaa kyselystä oikeat kohteet, myös XML-merkkikielellä. OpenLayers-kartta osaa lukea vastauksen ja muodostaa siitä kohteita kartalle kuvan 16 mukaisesti. Vastaus voi sisältää paljon tietoa kohteesta.



Kuva 16. Kohteita OpenLayers-kartalla.

6.4 Sovellusten välinen kommunikointi

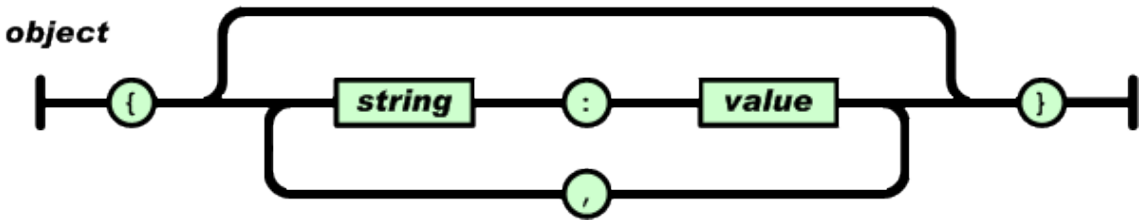
Karttapohjainen sopimuksenhallintajärjestelmä kommunikoi Geometrix Oy:n Työmaapäiväkirja-sovelluksen kanssa. Työmaapäiväkirja-sovelluksella pidetään kirjaa tehdyistä töistä, säätilasta, kalustosta ja työvoimasta. Karttapohjaisessa sopimuksenhallintajärjestelmässä halutaan näyttää Työmaapäiväkirja -sovelluksen käyttäjien tekemiä merkintöjä. Karttapohjainen sopimuksenhallintajärjestelmä vastaanottaa Työmaapäiväkirja-sovelluksesta JSON-muodossa lähetettyjä tietoja.

JSON on kevyt tiedonsiirtomuoto. Sen lukeminen ja kirjoittaminen on ihmiselle helppoa. Tietokoneet jäsentävät ja generoivat sitä helposti. JSON perustuu JavaScript-ohjelmointikielen ECMA-262 kolmannen painoksen joulukuun 1999 standardin osajoukkoon. JSON on tekstimuoto, joka on täysin ohjelmointikieliriippumaton, mutta se yleisiä tapoja, jotka ovat tuttuja C-kieliperheelle. JSON rakennetaan kahteen rakenteeseen:

- Kokoelmaan avain-arvopareja. Useissa kielissä tämä käsitetään oliona, luettelona, rakenteena, hakemistona, tiivisteenä, avainlistana tai assosiatiivisena ryhmänä.

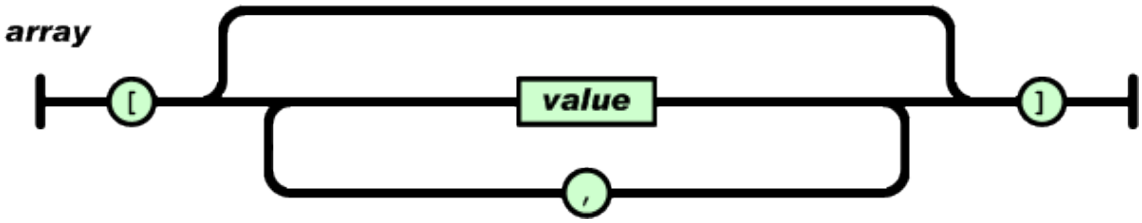
- Järjestettyyn listaan arvoja. Useimmissa ohjelmointikielissä tämä käsitetään ryhmänä, vektorina, listana tai sekvenssinä.

Nämä ovat universaaleja tietorakenteita ja yleensä modernit ohjelmointikielet tukevat näitä, joten on järkevää, että JSON on vaihdettavaa ohjelmointikielien kanssa, jotka tukevat myös näitä tietorakenteita.



Kuva 17. JSON-olion rakenne.

Kuvassa 17 nähdään kuvaus JSON-olion rakenteesta. Olio on joukko järjestämättömiä avain-arvopareja. Olio alkaa {-merkillä ja päättyy }-merkkiin. Jokainen nimi alkaa :-merkillä ja avain-arvoparit ovat eritelty ,-merkeillä.



Kuva 18. JSON-ryhmän rakenne.

Kuvassa 18 nähdään kuvaus JSON-ryhmän rakenteesta. Ryhmä on järjestetty joukko arvoja. Ryhmä alkaa [-merkillä ja päättyy]-merkkiin. Arvot erotellaan ,-merkillä. [35.].

Karttapohjainen sopimuksenhallintajärjestelmä vastaanottaa Työmaapäiväkirja-sovelluksen lähettämiä JSON-olioita ja jäsentelee ne. JSON-olioista muodostetaan Java-entity-olioita ja ne lisätään tietokantaan. Tietokannasta tiedot näytetään käyttäjälle JSP-sivulla.

7 Testaus

Yksikkötestaus

Sovelluksen testaus voidaan jakaa yksikkötestaukseen ja integraatiotestaukseen. Yksikkötestaus tehdään kooditasolla JUnitilla. JUnit on yksinkertainen viitekehys toistettavien testien kirjoittamiseen [30]. JUnitilla voidaan testata esimerkiksi tietokannan toimintaa tallentamalla ja hakemalla rivejä tietokannasta. Sovelluksessa on kirjoitettu yksikkötestejä monimutkaisille algoritmeille ja DAO-luokille. Hyvin kirjoitetut yksikkötestit löytävät nopeasti sovelluksen virheet ja nopeuttavat kehitystä. Esimerkiksi tietolomakkeen tallentamista voidaan nopeuttaa kirjoittamalla testi, joka lähettää tiedot servletille ilman, että kehittäjän tarvitsee aina käydä tietolomakkeen täyttämistä ja lähettämistä läpi jokaisen koodimuutoksen jälkeen.

```
/**
 * Test of calculateProductionPartQuality method, of class
 * QualityCalculator.
 */
@Test
public void testCalculateProductionPartQuality() {
    List<LassoFeatureDTO> lassoFeatureDTOs = createLassoFeatureDtos(exceptions);
    QualityCalculator instance = new QualityCalculator();
    double expResult = 2.25;
    double result = instance.calculateProductionPartQuality(instance.filter(lassoFeatureDTOs, "Lumen ja sohjon poisto"));
    assertEquals(expResult, result, 0.00);
}
```

Koodiesimerkki 8 JUnit testaus.

Koodiesimerkki 8 nähdään esimerkki JUnitilla tehdystä yksikkötestistä. Esimerkissä testattiin tuotanto-osan laadun laskemista. Testissä laatupoikkeamalistasta lasketaan QualityCalculator–luokan *calculateProductionPartQuality*-metodilla laatuarvo. Laatuarvoa verrataan *assertEquals*-metodilla arvoon 2.25. Arvon ollessa 2.25 testi menee läpi. Maven-plugin käy kaikki yksikkötestit läpi aina ohjelmaa kääntäessä, näin huomataan heti jos ohjelma ei toimi muutoksen jälkeen.

Integraatiotestaus

Integraatiotestauksessa testataan sovelluksen moduuleiden yhteensopivuutta ja vuorovaikutusta [31]. Sovelluksen perustoiminnot testataan nopeasti ennen uuden version julkaisemista. Tällä tavalla löydetään virheet ja voidaan todeta sovelluksen toimivaksi, jos virheitä ei löydy. Löydetyt virheet korjataan ja testitapaukset käydään uudelleen

läpi. Sovelluksesta kirjoitetaan testitapaukset, jotka kattavat perustoiminnot ja usein käytettävät toiminnot. Liitteessä 2 nähdään sovelluksen integraatiotestaussuunnitelma, jonka testaaja kävi läpi. Testaajan on hyvä olla henkilö, joka ei tiedä sovelluksesta paljon. Testaaja voi myös ottaa kantaa sovelluksen toimintoihin ja käytettävyyteen.

Ennen sovelluksen uuden version julkaisemista tehtävässä integraatiotestissä löydetään yleensä joitain virheitä, joita ohjelmoija ei ole huomannut tai yksikkötestit eivät kata. Testeissä yleisiä virheitä ovat kirjoitusvirheet ja toiminnallisuusvirheet käyttöliittymässä. Virheitä saattavat aiheuttaa myös tietokantamuutokset ja suuret uudelleenohjelmoinnit. Uudelleenohjelmointia kannattaa tehdä ajoittain, että koodi pysyy luettavana ja uudelleen käytettävänä vielä monien vuosien jälkeen.

8 Yhteenveto

Karttapohjaisesta sopimuksenhallintajärjestelmästä tuli lopulta hyvä työkalu sopimuksien tilanteiden seuraamiseen ja laadun tarkkailuun. Järjestelmän toimintoihin kuuluu esimerkiksi Helsingin auraus, suolaus ja hiekoitustilanteiden seuranta reaaliajassa ja urakka-alueiden työn laadun näkeminen kartalta yhdellä silmäyksellä.

Sovelluksen kehitys alkoi syksyllä 2013 ja sen pilottiversio saatiin valmiiksi aikataulussa joulukuun 2013 alussa. Sovelluksen kehittäminen jatkuu edelleen ja siihen lisätään vielä toimintoja, joita tilaaja näkee tarpeelliseksi.

Onnistumisena näen aikataulussa pysymisen ja sovelluksen ehjän kokonaisuuden, vaikka minulla ei ollut aikaisempaa kokemusta tämäntyyppisen web-sovelluksen kehittämisestä. Kehityksen aikana olen oppinut paljon uutta ja varmasti vielä opittavaa jäi. Käyttöliittymän hiomiseen ei jäänyt paljon aikaa, mutta siitä voidaan myöhemmin parannella, jos se koetaan tarpeelliseksi. Olisin mielelläni tutustunut kauemmin eri viitekehyksiin ja tekniikoihin, joilla sovelluksen olisi voinut tehdä. Palvelinpuolen olisi voinut toteuttaa REST-rajapintana, jotta käyttöliittymä ei olisi ollut riippuvainen JSP-tekniikasta.

Sovellus ei olisi tullut valmiiksi ilman työkavereideni avustusta ja opetusta. Erityisesti minua auttoi esimieheni ja insinöörityön ohjaajan Olli Alangon kirjoittamat dokumentit ja määrittelyt sekä hyvä projektisuunnitelma.

Lähteet

1. Suunnitteluprosessista. 2009. Verkkodokumentti. Virtuaali Ammattikorkeakoulu. <http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/030308/1146204519802/1146224777754/1146226151084/1146226538347.htm.l>. Luettu 13.10.2013.

2. Learn About Java Technology. 2013.. Verkkodokumentti. Oracle. <http://www.java.com/en/about/>. Luettu 13.10.2013.

3. About. Verkkodokumentti. PostgreSQL Global Development Group. <http://www.postgresql.org/about/>. Luettu 13.10.2013.

4. Ellen Siever, Stephen Figgins. 2009. Linux in a Nutshell. O'Reilly Media

5. Laakkonen Sami. 2009. Ajax-verkkosovellukset: Timeo ajanseurantasovellus. Insinöörityö Metropolia Ammattikorkeakoulu.

6. What is jQuery. 2013. Verkkodokumentti. jQuery. <http://jquery.com/>. Luettu 28.10.2013.

7. jQueryUI. 2013. Verkkodokumentti. jQueryUI. <http://jqueryui.com/>. Luettu 28.10.2013.

8. Documentation. 2013. Verkkodokumentti. tablesorter. <http://tablesorter.com/docs/>. Luettu 28.10.2013.

9. NetBeans IDE - The Smarter and Faster Way to Code. 2013. Verkkodokumentti. NetBeans. <https://netbeans.org/features/index.html>. Luettu 28.10.2013.

10. What is maven. 2013. Verkkodokumentti. Apache 2013. <http://maven.apache.org/what-is-maven.html>. Luettu 28.10.2013.

11. Introduction to the pom. 2013. Verkkodokumentti. <http://maven.apache.org/guides/introduction/introduction-to-the-pom.html#Minimal_POM>. Luettu 28.10.2013.

12. JDBC. 2013. Verkkodokumentti. Webopedia <http://www.webopedia.com/TERM/J/JDBC.html>. Luettu 28.10.2013.

13. Oracle 2013, verkkodokumentti <http://docs.oracle.com/cd/E19879-01/820-4335/ablil/index.html>. Luettu 28.10.2013.

14. How JDBC Resources and Connection Pools Work Together. 2013. Verkkodokumentti. Oracle <http://www.oracle.com/technetwork/java/overview-142035.html>. Luettu 15.11.2013.

15. Documentation. 2013. Verkkodokumentti. OpenLayers. <http://docs.openlayers.org/>. Luettu 15.11.2013.

16. OpenStreetMap. 2013. Verkkodokumentti. OpenStreetMap <http://www.openstreetmap.org>. Luettu 15.11.2013.

17. PostGIS. 2013. Verkkodokumentti. PostGIS <http://postgis.net/>. Luettu 15.11.2013.

18. WMS. 2013. Verkkodokumentti. OGC. <http://www.opengeospatial.org/standards/wms>. Luettu 15.11.2013.

19. WFS. 2013. Verkkodokumentti. OGC. 2013. <http://www.opengeospatial.org/standards/wfs>. Luettu 15.11.2013.

20. MVC patterns. 2013. Verkkodokumentti. IBM <http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.ajax.devguide.help%2Fdocs%2FPureAjax_MVC_patterns.html>. Luettu 15.11.2013.

21. Annotations. 2013. Verkkodokumentti. Oracle. <http://docs.oracle.com/javase/tutorial/java/annotations/ >. Luettu 29.11.2013.

22. Entities. Verkkodokumentti. 2013. Oracle <http://docs.oracle.com/javaee/5/tutorial/doc/bnbqa.html>. Luettu 29.11.2013.

23. JavaServer Pages Overview. Verkkodokumentti 2013. Oracle. <http://www.oracle.com/technetwork/java/overview-138580.html>.Luettu 29.11.2013.

24. Javadokumenttien kirjoittaminen - JavaDoc . 2013. Verkkodokumentit. Tietojenkäsittelytieteen laitos. <http://www.cs.helsinki.fi/node/61338>. Luettu 29.11.2013.

25. Mockingbird. Verkkodokumentti. 2013. Mockingbird. <https://gomockingbird.com/>. Luettu 29.11.2013.

26. What is hibernate. 2005. Verkkodokumentti. James Elliot. <http://www.onjava.com/pub/a/onjava/2005/09/21/what-is-hibernate.html>. Luettu 29.11.2013.

27. Java Servlet Technology Overview . 2013. Oracle. <http://www.oracle.com/technetwork/java/overview-137084.html>. Luettu 30.11.2013.

28. Strategy pattern. 2013. Verkkodokumentti. OODesign. <http://www.oodesign.com/strategy-pattern.html>. Luettu 30.11.2013.

29. Factory-pattern. 2013. Verkkodokumentti. OODesign. <http://www.oodesign.com/factory-pattern.html>. Luettu 30.11.2013.

30. JUnit. 2013, Verkkodokumentti. GitHub. <https://github.com/junit-team/junit>, Luettu 08.12.2013.

31. Menetelmädokumentti: Integraatiotestaus. 2003. Verkkodokumentti. SoberIT. <http://www.soberit.hut.fi/T-76.115/02-03/palautukset/groups/Jove/de/documents/method_descriptions/method_description_AK.html>, Luettu 08.12.2013.

32. Versionhallinta. 2013. Verkkodokumentti. TietoWeb. <http://www.tietoweb.fi/artikkelit/versionhallinta>, Luettu 10.12.2013.

33. Jakob Nielsen. 2006. Prioritizing web usability.

34. Jakob Nielsen. 2000. Designing Web Usability..

35. Introducing JSON. Verkkodokumentti. JSON. <http://www.json.org/ >, Luettu 14.12.2013.

Sopimuksen tilan katselu käyttötapaus

Käyttötapaus	**SOPIMUKSEN TILAN KATSELU**
Yhteenveto	Käyttäjä tarkistaa sopimuksen laadullisen tilan
Tekijät	Osastopäälliköt, päättäjät, ylläpidon viranhaltijat, tilaajat, tuottajat, valvojat.
Esiehdot	
Kuvaus	Käyttäjä hakee haluamansa sopimuksen tiedot esiin hakemalla sopimuksen tiedot esiin käyttötapauksen Tietojen hakeminen mukaisesti. Sovellus näyttää hakutuloksena hakukriteereihin sopivat sopimukset, joista käyttäjä valitsee haluamansa yksi kerrallaan. Sopimuksen tiedot tulevat näkyviin sopimusnäytölle ja käyttäjä voi katsella sopimuksen senhetkistä tilaa. Perustietolomakkeella näytetään sopimuksesta seuraavat tiedot: - nimi - kuvaus - diaarinumero - viite (sopimushallintajärjestelmään) - alku- ja loppupvm - urakoitsija Lisätietolomakkeilla on välilehtien alla nähtävillä sopimukseen liittyviä liitetietoja seuraavasti: - kommentit viestiketjuna - dokumentit ryhmiteltynä omiin kansioihinsa - raportit - laatutiedot (laatupoikkeamat+poikkeamaraportit, laatumittaritiedot, johdetut/lasketut laatuarvot) - työmaapäiväkirjat

	- tapahtumat (järjestelmän kirjaamat sopimukseen liittyvät tapahtumat) - sopimuksen henkilöt - tuotteet ja tuotanto-osat - kartta (mikäli ei tehdä Lasson yhteistä jostakin painikkeesta avautuvaa kartta) Lisätietolomakkeilta voi siirtyä katsomaan tarkemmin jonkin lisätiedon tietoja seuraavasti: - valitsemalla jonkin työmaapäiväkirjan siirtyy sovellus Työmaapäiväkirjaosioon siten, että aktiivisena on äsken valittu työmaapäiväkirja, ja hakutuloslistassa ovat selattavina äsken käsitellyn sopimuksen työmaapäiväkirjat - valitsemalla jokin laatutieto siirtyy sovellus Laatutiedot-osioon siten, että aktiivisena on äsken valittu laatutieto, ja hakutuloslistassa ovat selattavina äsken käsitellyn sopimuksen laatutiedot - valitsemalla jokin tuote tai tuotanto-osa siirtyy sovellus Tuotteet-osioon siten, että aktiivisena on äsken valittu tuote (tuotekortti), ja hakutuloslistassa ovat selattavina äsken käsitellyn sopimuksen tuotteet Käyttäjä voi avata karttalomakkeen erillisestä painikkeesta, ja tällöin avautuu karttanäkymä, joka zoomautuu siten, että sopimuksen rajat ovat näkyvissä. Käyttäjä voi hakea karttanäkymään sopimuksen tilaa kuvaavia karttaesitystietoja avaamalla seuraavia karttatasoja: - asukaspalautteet sopimuksen mukaisen vastuulajin mukaan halutulta aikaväliltä teemoitettuna väreillä sen mukaan ovatko ne auki/suljettuna - tilannekuva, mikäli mahdollista (esim. talvihoidon tilannekuva on saatavilla) - laatuväri-indikaattori kartalle (indikaattorin väri syntyy heuristisesti järjestelmän tekemän analyysin perusteella, oletuksena vihreä.) - laatupoikkeamat halutulta aikaväliltä sopimuksen mukaisen tuotanto-osan mukaan automaattisesti haettuna - erilaiset taustakartta-aineistot, kuten kantakartta, kiinteistökartta, ilmakuva

	Käyttäjä voi kommentoida sopimusta kirjoittamalla jotakin kommenttikenttään tai vastaamalla johonkin aiemmin kirjattuun kommenttiin.
Poikkeukset	Käyttäjällä oltava vähintään lukuoikeus sopimuksen tietoihin.
Lopputulos	Käyttäjä näkee tietoja katselemalla sopimuksen tilanteen työsuoritusten ja laatutietojen valossa.

Luokkien vastuut	**Yhteistyö** Tietojen hakeminen

Sovelluksen integraatiotestaus

Lasso integraatiotestaus
Verkkosovellus

Versio:
Testaaja:

nro	tapaus	kuvaus	odotettu lopputulos	testattu	tulos
	YLEISET				
1000	Kirjautuminen	Kirjaudu sovellukseen käyttäjällä testi/testi	Sovelluksen etusivu aukeaa (tyhjä)		
1001	Takaisin-painike	Testataan selaimen historian toimivuutta	Selaimen takaisin ja eteenpäin painikkeet toimivat		
1002	URL testaaminen	Sovelluksessa kokeillaan URL osoitteilla navigointia.	Selain lataa esimerkiksi oikean sopimuksen kun sopimuksen URL osoite ladataan.		
1003	Listan suurentaminen ja pienentäminen	Suurennetaan ja pienennetään listaa sopimuksen, tuotekortin, urakoitsijan ja työmaapäiväkirjan tietolomakkeella	Toiminto toimii ja toimii järkevästi.		
	SOPIMUKSET				
2000	Sopimuksen luonti	Luodaan uusi sopimus sopimuksen tietolomakkeelta	Sopimus tallentuu tietokantaan ja sopimuksen tiedot ovat oikein uudelleen ladattaessa. Sopimuksen tapahtumat välilehteen ilmestyy sopimuksen luonnista merkintä		
2001	Sopimukseen henkilön lisääminen	Lisätään sopimukseen henkilö: Miikka Salomaa	Henkilön lisääminen onnistuu ja sopimuksen henkilöä voi muokata ja sen voi poistaa.		
2002	Sopimuksen muokkaaminen	Muokataan sopimuksen tietoja.	Muokkaaminen onnistuu normaalisti. Muokkaamisesta jää merkintä sopimuksen tapahtumiin.		
2003	Sopimuksen kommentit	Sopimukseen voi lisätä kommentteja ja kommentteihin voi viitata kommentilla	Kommentit näkyvät sopimuksessa normaalisti ja kaikki tiedot ovat oikein.		
2004	Sopimusten haku	Toistetaan testi 2000 ja haetaan sopimuksia eri yhdistelmillä	Sopimuksen hakukriteerit vastaavat hakutuloksia.		
2005	Sopimusten tuotteet	Lisätään sopimukseen	Sopimuksen tuotteen		

		tuotteita	lisääminen onnistuu
2006	Kartta	Avataan sopimuksen ”kartta” -välilehti	Kartta toimii. Ei näytä vielä mitään lassoon liittyvää dataa
	Tuotekortit		
3000	Sopimus tuotekortissa	Äsken lisätty sopimus näkyy tuotekortin tiedossa	Sopimusta klikkaamalla pääsee sopimuksen tietoihin ja sopimuksen tiedoista takaisin tuotekorttiin.
3001	Tuotekortin tietojen muokkaaminen	Tuotekortin tietoja voidaan muokata tietolomakkeelta	Tietoja voidaan muokata ja tiedot ovat oikein.
3002	Tuotekortin kommentit	Tuotekorttia voi kommentoida ja kommentteihin voidaan viitata kommentilla	Kommentit näkyvät tuotekortissa normaalisti ja kaikki tiedot ovat oikein.
	Urakoitsijat		
4001	Urakoitsijan luonti	Luodaan uusi urakoitsija tietolomakkeelta	Urakoitsija tallentuu tietokantaan ja tiedot ovat oikein.
4002	Sopimuksen liittäminen urakoitsijaan	Muutetaan aikaisemmin luodun sopimuksen urakoitsijaksi äsken luotu urakoitsija	Urakoitsijan vaihtaminen onnistuu ja urakoitsijan ”sopimukset” -välilehdessä näkyy oikea sopimus ja sitä klikkaamalla päästää sopimukseen ja sopimuksesta urakoitsija-linkistä urakoitsijan tietoihin.
4003	Urakoitsijan tietojen muokkaaminen	Muokataan urakoitsijan tietoja	Tietojen muokkaaminen onnistuu ja tiedot tallentuvat tietokantaan oikein.
	Työmaapäiväkirjat		
5000	Työmaapäiväkirjan hakeminen	Haetaan työmaapäiväkirjoja eri hakukriteereillä	Hakutulokset vastaavat hakukriteereitä.
5001	Navigointi	Työmaapäiväkirjasta voidaan navigoida linkin kautta helposti sopimukseen, urakoitsijaan ja tuotteeseen	Siirtymät toimivat ja oikeat tiedot aukeavat.